Arm® Architecture Registers
Future Architecture Technologies in the A architecture profile

Beta
AArch64 System Registers

**ACTLR_EL1**: Auxiliary Control Register (EL1)

**ACTLR_EL2**: Auxiliary Control Register (EL2)

**ACTLR_EL3**: Auxiliary Control Register (EL3)

**AFSR0_EL1**: Auxiliary Fault Status Register 0 (EL1)

**AFSR0_EL2**: Auxiliary Fault Status Register 0 (EL2)

**AFSR0_EL3**: Auxiliary Fault Status Register 0 (EL3)

**AFSR1_EL1**: Auxiliary Fault Status Register 1 (EL1)

**AFSR1_EL2**: Auxiliary Fault Status Register 1 (EL2)

**AFSR1_EL3**: Auxiliary Fault Status Register 1 (EL3)

**AIDR_EL1**: Auxiliary ID Register

**AMAIR_EL1**: Auxiliary Memory Attribute Indirection Register (EL1)

**AMAIR_EL2**: Auxiliary Memory Attribute Indirection Register (EL2)

**AMAIR_EL3**: Auxiliary Memory Attribute Indirection Register (EL3)

**AMCFGR_EL0**: Activity Monitors Configuration Register

**AMCG1IDR_EL0**: Activity Monitors Counter Group 1 Identification Register

**AMCGCR_EL0**: Activity Monitors Counter Group Configuration Register

**AMCNTENCLR0_EL0**: Activity Monitors Count Enable Clear Register 0

**AMCNTENCLR1_EL0**: Activity Monitors Count Enable Clear Register 1

**AMCNTENSET0_EL0**: Activity Monitors Count Enable Set Register 0

**AMCNTENSET1_EL0**: Activity Monitors Count Enable Set Register 1

**AMCR_EL0**: Activity Monitors Control Register

**AMEVCNTR0<n>_EL0**: Activity Monitors Event Counter Registers 0

**AMEVCNTR1<n>_EL0**: Activity Monitors Event Counter Registers 1

**AMEVCNTVOFF0<n>_EL2**: Activity Monitors Event Counter Virtual Offset Registers 0

**AMEVCNTVOFF1<n>_EL2**: Activity Monitors Event Counter Virtual Offset Registers 1

**AMEVTYPER0<n>_EL0**: Activity Monitors Event Type Registers 0

**AMEVTYPER1<n>_EL0**: Activity Monitors Event Type Registers 1

**AMUSERENR_EL0**: Activity Monitors User Enable Register

**APDAKeyHi_EL1**: Pointer Authentication Key A for Data (bits[127:64])

**APDAKeyLo_EL1**: Pointer Authentication Key A for Data (bits[63:0])

**APDBKeyHi_EL1**: Pointer Authentication Key B for Data (bits[127:64])

**APDBKeyLo_EL1**: Pointer Authentication Key B for Data (bits[63:0])

**APGAKeyHi_EL1**: Pointer Authentication Key A for Code (bits[127:64])

**APGAKeyLo_EL1**: Pointer Authentication Key A for Code (bits[63:0])
AArch64 System Registers

APIAKeyHi_EL1: Pointer Authentication Key A for Instruction (bits[127:64])
APIAKeyLo_EL1: Pointer Authentication Key A for Instruction (bits[63:0])
APIBKeyHi_EL1: Pointer Authentication Key B for Instruction (bits[127:64])
APIBKeyLo_EL1: Pointer Authentication Key B for Instruction (bits[63:0])
CCSIDR2_EL1: Current Cache Size ID Register 2
CCSIDR_EL1: Current Cache Size ID Register
CLIDR_EL1: Cache Level ID Register
CNTFRO_EL0: Counter-timer Frequency register
CNTHCTL_EL2: Counter-timer Hypervisor Control register
CNTHPS_CTL_EL2: Counter-timer Secure Physical Timer Control register (EL2)
CNTHPS_CVAL_EL2: Counter-timer Secure Physical Timer CompareValue register (EL2)
CNTHPS_TVAL_EL2: Counter-timer Secure Physical Timer TimerValue register (EL2)
CNTHP_CTL_EL2: Counter-timer Hypervisor Physical Timer Control register
CNTHP_CVAL_EL2: Counter-timer Physical Timer CompareValue register (EL2)
CNTHP_TVAL_EL2: Counter-timer Physical Timer TimerValue register (EL2)
CNTHSV_CTL_EL2: Counter-timer Secure Virtual Timer Control register (EL2)
CNTHSV_CVAL_EL2: Counter-timer Secure Virtual Timer CompareValue register (EL2)
CNTHSV_TVAL_EL2: Counter-timer Secure Virtual Timer TimerValue register (EL2)
CNTHV_CTL_EL2: Counter-timer Virtual Timer Control register (EL2)
CNTHV_CVAL_EL2: Counter-timer Virtual Timer CompareValue register (EL2)
CNTHV_TVAL_EL2: Counter-timer Virtual Timer TimerValue Register (EL2)
CNTKCTL_EL1: Counter-timer Kernel Control register
CNTPCSS_EL0: Counter-timer Self-Synchronized Physical Count register
CNTPCTL_EL0: Counter-timer Physical Count register
CNTPOFF_EL2: Counter-timer Physical Offset register
CNTPS_CTL_EL1: Counter-timer Physical Secure Timer Control register
CNTPS_CVAL_EL1: Counter-timer Physical Secure Timer CompareValue register
CNTPS_TVAL_EL1: Counter-timer Physical Secure Timer TimerValue register
CNTP_CTL_EL0: Counter-timer Physical Timer Control register
CNTP_CVAL_EL0: Counter-timer Physical Timer CompareValue register
CNTP_TVAL_EL0: Counter-timer Physical Timer TimerValue register
CNTVCSS_EL0: Counter-timer Self-Synchronized Virtual Count register
CNTVCT_EL0: Counter-timer Virtual Count register
CNTVOFF_EL2: Counter-timer Virtual Offset register
CNTV_CTL_EL0: Counter-timer Virtual Timer Control register
CNTV_CVAL_EL0: Counter-timer Virtual Timer CompareValue register
COUNTV_TVAL_EL0: Counter-timer Virtual Timer TimerValue register

CONTEXTIDR_EL1: Context ID Register (EL1)

CONTEXTIDR_EL2: Context ID Register (EL2)

CPACR_EL1: Architectural Feature Access Control Register

CPTR_EL2: Architectural Feature Trap Register (EL2)

CPTR_EL3: Architectural Feature Trap Register (EL3)

CSSELR_EL1: Cache Size Selection Register

CTR_EL0: Cache Type Register

CurrentEL: Current Exception Level

DACR32_EL2: Domain Access Control Register

DAIF: Interrupt Mask Bits

DBGAUTHSTATUS_EL1: Debug Authentication Status register

DBGBCR<n>_EL1: Debug Breakpoint Control Registers

DBGBVR<n>_EL1: Debug Breakpoint Value Registers

DBGCLAIMCLR_EL1: Debug CLAIM Tag Clear register

DBGCLAIMSET_EL1: Debug CLAIM Tag Set register

DBGDTRR<EL0: Debug Data Transfer Register, Receive

DBGDTTX<EL0: Debug Data Transfer Register, Transmit

DBGDTR_EL0: Debug Data Transfer Register, half-duplex

DBGPRCR_EL1: Debug Power Control Register

DBGVCR32_EL2: Debug Vector Catch Register

DBGWCR<n> EL1: Debug Watchpoint Control Registers

DBGWVR<n> EL1: Debug Watchpoint Value Registers

DCZID_EL0: Data Cache Zero ID register

DISR_EL1: Deferred Interrupt Status Register

DIT: Data Independent Timing

DLR_EL0: Debug Link Register

DSPSR_EL0: Debug Saved Program Status Register

ELR_EL1: Exception Link Register (EL1)

ELR_EL2: Exception Link Register (EL2)

ELR_EL3: Exception Link Register (EL3)

ERRIDR_EL1: Error Record ID Register

ERRSELR_EL1: Error Record Select Register

ERXADDR_EL1: Selected Error Record Address Register

ERXCTLR_EL1: Selected Error Record Control Register

ERXFR_EL1: Selected Error Record Feature Register
ERXMISC0_EL1: Selected Error Record Miscellaneous Register 0
ERXMISC1_EL1: Selected Error Record Miscellaneous Register 1
ERXMISC2_EL1: Selected Error Record Miscellaneous Register 2
ERXMISC3_EL1: Selected Error Record Miscellaneous Register 3
ERXPFGCDN_EL1: Selected Pseudo-fault Generation Countdown Register
ERXPFGCTL_EL1: Selected Pseudo-fault Generation Control Register
ERXPFGF_EL1: Selected Pseudo-fault Generation Feature Register
ERXSTATUS_EL1: Selected Error Record Primary Status Register
ESR_EL1: Exception Syndrome Register (EL1)
ESR_EL2: Exception Syndrome Register (EL2)
ESR_EL3: Exception Syndrome Register (EL3)
FAR_EL1: Fault Address Register (EL1)
FAR_EL2: Fault Address Register (EL2)
FAR_EL3: Fault Address Register (EL3)
FPCR: Floating-point Control Register
FPEXC32_EL2: Floating-Point Exception Control register
FPSR: Floating-point Status Register
GCR_EL1: Tag Control Register.
GMID_EL1: Multiple tag transfer ID register
HACR_EL2: Hypervisor Auxiliary Control Register
HAFGRTR_EL2: Hypervisor Activity Monitors Fine-Grained Read Trap Register
HCR_EL2: Hypervisor Configuration Register
HDFGRTR_EL2: Hypervisor Debug Fine-Grained Read Trap Register
HDFGWTR_EL2: Hypervisor Debug Fine-Grained Write Trap Register
HFGITR_EL2: Hypervisor Fine-Grained Instruction Trap Register
HFGRTR_EL2: Hypervisor Fine-Grained Read Trap Register
HFGWTR_EL2: Hypervisor Fine-Grained Write Trap Register
HPFAR_EL2: Hypervisor IPA Fault Address Register
HSTR_EL2: Hypervisor System Trap Register
ICC_AP0R<n>_EL1: Interrupt Controller Active Priorities Group 0 Registers
ICC_AP1R<n>_EL1: Interrupt Controller Active Priorities Group 1 Registers
ICC_ASGI1R_EL1: Interrupt Controller Alias Software Generated Interrupt Group 1 Register
ICC_BPR0_EL1: Interrupt Controller Binary Point Register 0
ICC_BPR1_EL1: Interrupt Controller Binary Point Register 1
ICC_CTLR_EL1: Interrupt Controller Control Register (EL1)
ICC_CTLR_EL3: Interrupt Controller Control Register (EL3)
ICC_DIR_EL1: Interrupt Controller Deactivate Interrupt Register
ICC_EOIR0_EL1: Interrupt Controller End Of Interrupt Register 0
ICC_EOIR1_EL1: Interrupt Controller End Of Interrupt Register 1
ICC_HPPIR0_EL1: Interrupt Controller Highest Priority Pending Interrupt Register 0
ICC_HPPIR1_EL1: Interrupt Controller Highest Priority Pending Interrupt Register 1
ICC_IAR0_EL1: Interrupt Controller Interrupt Acknowledge Register 0
ICC_IAR1_EL1: Interrupt Controller Interrupt Acknowledge Register 1
ICC_IGRPEN0_EL1: Interrupt Controller Interrupt Group 0 Enable register
ICC_IGRPEN1_EL1: Interrupt Controller Interrupt Group 1 Enable register
ICC_IGRPEN1_EL3: Interrupt Controller Interrupt Group 1 Enable register (EL3)
ICC_PMR_EL1: Interrupt Controller Interrupt Priority Mask Register
ICC_RPR_EL1: Interrupt Controller Running Priority Register
ICC_SGI0R_EL1: Interrupt Controller Software Generated Interrupt Group 0 Register
ICC_SGI1R_EL1: Interrupt Controller Software Generated Interrupt Group 1 Register
ICC_SRE_EL1: Interrupt Controller System Register Enable register (EL1)
ICC_SRE_EL2: Interrupt Controller System Register Enable register (EL2)
ICC_SRE_EL3: Interrupt Controller System Register Enable register (EL3)
ICH_AP0R<n>_EL2: Interrupt Controller Hyp Active Priorities Group 0 Registers
ICH_AP1R<n>_EL2: Interrupt Controller Hyp Active Priorities Group 1 Registers
ICH_EISR_EL2: Interrupt Controller End of Interrupt Status Register
ICH_ELRSR_EL2: Interrupt Controller Empty List Register Status Register
ICH_HCR_EL2: Interrupt Controller Hyp Control Register
ICH_LR<n>_EL2: Interrupt Controller List Registers
ICH_MISR_EL2: Interrupt Controller Maintenance Interrupt State Register
ICH_VMCR_EL2: Interrupt Controller Virtual Machine Control Register
ICH_VTR_EL2: Interrupt Controller VGIC Type Register
ICV_AP0R<n>_EL1: Interrupt Controller Virtual Active Priorities Group 0 Registers
ICV_AP1R<n>_EL1: Interrupt Controller Virtual Active Priorities Group 1 Registers
ICV_BPR0_EL1: Interrupt Controller Virtual Binary Point Register 0
ICV_BPR1_EL1: Interrupt Controller Virtual Binary Point Register 1
ICV_CTRL_EL1: Interrupt Controller Virtual Control Register
ICV_DIR_EL1: Interrupt Controller Deactivate Virtual Interrupt Register
ICV_EOIR0_EL1: Interrupt Controller Virtual End Of Interrupt Register 0
ICV_EOIR1_EL1: Interrupt Controller Virtual End Of Interrupt Register 1
ICV_HPPIR0_EL1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
ICV_HPPIR1_EL1: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
ICV_IAR0_EL1: Interrupt Controller Virtual Interrupt Acknowledge Register 0
ICV_IAR1_EL1: Interrupt Controller Virtual Interrupt Acknowledge Register 1
ICV_IGRPEN0_EL1: Interrupt Controller Virtual Interrupt Group 0 Enable register
ICV_IGRPEN1_EL1: Interrupt Controller Virtual Interrupt Group 1 Enable register
ICV_PMR_EL1: Interrupt Controller Virtual Interrupt Priority Mask Register
ICV_RPR_EL1: Interrupt Controller Virtual Running Priority Register
ID_AA64AFR0_EL1: AArch64 Auxiliary Feature Register 0
ID_AA64AFR1_EL1: AArch64 Auxiliary Feature Register 1
ID_AA64DFR0_EL1: AArch64 Debug Feature Register 0
ID_AA64DFR1_EL1: AArch64 Debug Feature Register 1
ID_AA64ISAR0_EL1: AArch64 Instruction Set Attribute Register 0
ID_AA64ISAR1_EL1: AArch64 Instruction Set Attribute Register 1
ID_AA64MMFR0_EL1: AArch64 Memory Model Feature Register 0
ID_AA64MMFR1_EL1: AArch64 Memory Model Feature Register 1
ID_AA64MMFR2_EL1: AArch64 Memory Model Feature Register 2
ID_AA64PFR0_EL1: AArch64 Processor Feature Register 0
ID_AA64PFR1_EL1: AArch64 Processor Feature Register 1
ID_AA64ZFR0_EL1: SVE Feature ID register 0
ID_AFR0_EL1: AArch32 Auxiliary Feature Register 0
ID_DFR0_EL1: AArch32 Debug Feature Register 0
ID_DFR1_EL1: Debug Feature Register 1
ID_ISAR0_EL1: AArch32 Instruction Set Attribute Register 0
ID_ISAR1_EL1: AArch32 Instruction Set Attribute Register 1
ID_ISAR2_EL1: AArch32 Instruction Set Attribute Register 2
ID_ISAR3_EL1: AArch32 Instruction Set Attribute Register 3
ID_ISAR4_EL1: AArch32 Instruction Set Attribute Register 4
ID_ISAR5_EL1: AArch32 Instruction Set Attribute Register 5
ID_ISAR6_EL1: AArch32 Instruction Set Attribute Register 6
ID_MMFR0_EL1: AArch32 Memory Model Feature Register 0
ID_MMFR1_EL1: AArch32 Memory Model Feature Register 1
ID_MMFR2_EL1: AArch32 Memory Model Feature Register 2
ID_MMFR3_EL1: AArch32 Memory Model Feature Register 3
ID_MMFR4_EL1: AArch32 Memory Model Feature Register 4
ID_MMFR5_EL1: AArch32 Memory Model Feature Register 5
ID_PFR0_EL1: AArch32 Processor Feature Register 0
ID_PFR1_EL1: AArch32 Processor Feature Register 1
AArch64 System Registers

ID_PFR2_EL1: AArch32 Processor Feature Register 2
IFSR32_EL2: Instruction Fault Status Register (EL2)
ISR_EL1: Interrupt Status Register
LORC_EL1: LORegion Control (EL1)
LOREA_EL1: LORegion End Address (EL1)
LORID_EL1: LORegionID (EL1)
LORN_EL1: LORegion Number (EL1)
LORSA_EL1: LORegion Start Address (EL1)
MAIR_EL1: Memory Attribute Indirection Register (EL1)
MAIR_EL2: Memory Attribute Indirection Register (EL2)
MAIR_EL3: Memory Attribute Indirection Register (EL3)
MDCCINT_EL1: Monitor DCC Interrupt Enable Register
MDCCSR_EL0: Monitor DCC Status Register
MDCR_EL2: Monitor Debug Configuration Register (EL2)
MDCR_EL3: Monitor Debug Configuration Register (EL3)
MDRAR_EL1: Monitor Debug ROM Address Register
MDSCR_EL1: Monitor Debug System Control Register
MIDR_EL1: Main ID Register
MPAM0_EL1: MPAM0 Register (EL1)
MPAM1_EL1: MPAM1 Register (EL1)
MPAM2_EL2: MPAM2 Register (EL2)
MPAM3_EL3: MPAM3 Register (EL3)
MPAMHCR_EL2: MPAM Hypervisor Control Register (EL2)
MPAMIDR_EL1: MPAM ID Register (EL1)
MPAMVP0_EL2: MPAM Virtual PARTID Mapping Register 0
MPAMVP1_EL2: MPAM Virtual PARTID Mapping Register 1
MPAMVP2_EL2: MPAM Virtual PARTID Mapping Register 2
MPAMVP3_EL2: MPAM Virtual PARTID Mapping Register 3
MPAMVP4_EL2: MPAM Virtual PARTID Mapping Register 4
MPAMVP5_EL2: MPAM Virtual PARTID Mapping Register 5
MPAMVP6_EL2: MPAM Virtual PARTID Mapping Register 6
MPAMVP7_EL2: MPAM Virtual PARTID Mapping Register 7
MPAMVPMV_EL2: MPAM Virtual Partition Mapping Valid Register
MPIDR_EL1: Multiprocessor Affinity Register
MVFR0_EL1: AArch32 Media and VFP Feature Register 0
MVFR1_EL1: AArch32 Media and VFP Feature Register 1
MVFR2_EL1: AArch32 Media and VFP Feature Register 2
NZCV: Condition Flags
OSDLR_EL1: OS Double Lock Register
OSDTRRX_EL1: OS Lock Data Transfer Register, Receive
OSDTRTX_EL1: OS Lock Data Transfer Register, Transmit
OSECCR_EL1: OS Lock Exception Catch Control Register
OSLAR_EL1: OS Lock Access Register
OSLSR_EL1: OS Lock Status Register
PAN: Privileged Access Never
PAR_EL1: Physical Address Register
PMBIDR_EL1: Profiling Buffer ID Register
PMBLIMITR_EL1: Profiling Buffer Limit Address Register
PMBPTR_EL1: Profiling Buffer Write Pointer Register
PMBSR_EL1: Profiling Buffer Status/syndrome Register
PMCCFILTR_EL0: Performance Monitors Cycle Count Filter Register
PMCCNTR_EL0: Performance Monitors Cycle Count Register
PMCEID0_EL0: Performance Monitors Common Event Identification register 0
PMCEID1_EL0: Performance Monitors Common Event Identification register 1
PMCNTENCLR_EL0: Performance Monitors Count Enable Clear register
PMCNTENSET_EL0: Performance Monitors Count Enable Set register
PMCR_EL0: Performance Monitors Control Register
PMEVCNTR<n>_EL0: Performance Monitors Event Count Registers
PMEVTYPER<n>_EL0: Performance Monitors Event Type Registers
PMINTENCLR_EL1: Performance Monitors Interrupt Enable Clear register
PMINTENSET_EL1: Performance Monitors Interrupt Enable Set register
PMMIR_EL1: Performance Monitors Machine Identification Register
PMOVSCLR_EL0: Performance Monitors Overflow Flag Status Clear Register
PMOVSET_EL0: Performance Monitors Overflow Flag Status Set register
PMSCR_EL1: Statistical Profiling Control Register (EL1)
PMSCR_EL2: Statistical Profiling Control Register (EL2)
PMSELR_EL0: Performance Monitors Event Counter Selection Register
PMSEVFR_EL1: Sampling Event Filter Register
PMSFCCR_EL1: Sampling Filter Control Register
PMSICR_EL1: Sampling Interval Counter Register
PMSIDR_EL1: Sampling Profiling ID Register
PMSIRR_EL1: Sampling Interval Reload Register
Sampling Latency Filter Register

Performance Monitors Software Increment Register

Performance Monitors User Enable Register

Performance Monitors Selected Event Count Register

Performance Monitors Selected Event Type Register

Revision ID Register

Random Allocation Tag Seed Register

Reset Management Register (EL1)

Reset Management Register (EL2)

Reset Management Register (EL3)

Random Number

Reseeded Random Number

Reset Vector Base Address Register (if EL2 and EL3 not implemented)

Reset Vector Base Address Register (if EL3 not implemented)

Reset Vector Base Address Register (if EL3 implemented)

IMPLEMENTATION DEFINED registers

Secure Configuration Register

System Control Register (EL1)

System Control Register (EL2)

System Control Register (EL3)

EL0 Read/Write Software Context Number

EL1 Read/Write Software Context Number

EL2 Read/Write Software Context Number

EL3 Read/Write Software Context Number

AArch32 Secure Debug Enable Register

AArch32 Secure Debug Enable Register

Saved Program Status Register (EL1)

Saved Program Status Register (EL2)

Saved Program Status Register (EL3)

Saved Program Status Register (Abort mode)

Saved Program Status Register (FIQ mode)

Saved Program Status Register (IRQ mode)

Saved Program Status Register (Undefined mode)

Stack Pointer Select

Stack Pointer (EL0)

Stack Pointer (EL1)
**SP_EL2**: Stack Pointer (EL2)

**SP_EL3**: Stack Pointer (EL3)

**SSBS**: Speculative Store Bypass Safe

**TCO**: Tag Check Override

**TCR_EL1**: Translation Control Register (EL1)

**TCR_EL2**: Translation Control Register (EL2)

**TCR_EL3**: Translation Control Register (EL3)

**TFSRE0_EL1**: Tag Fail Status Register (EL0).

**TFSR_EL1**: Tag Fail Status Register (EL1)

**TFSR_EL2**: Tag Fail Status Register (EL2)

**TFSR_EL3**: Tag Fail Status Register (EL3)

**TPIDRRO_EL0**: EL0 Read-Only Software Thread ID Register

**TPIDR_EL0**: EL0 Read/Write Software Thread ID Register

**TPIDR_EL1**: EL1 Software Thread ID Register

**TPIDR_EL2**: EL2 Software Thread ID Register

**TPIDR_EL3**: EL3 Software Thread ID Register

**TRBBASER_EL1**: Trace Buffer Base Address Register

**TRBIDR_EL1**: Trace Buffer ID Register

**TRBLIMITR_EL1**: Trace Buffer Limit Address Register

**TRBMAR_EL1**: Trace Buffer Memory Attribute Register

**TRBPTR_EL1**: Trace Buffer Write Pointer Register

**TRBSR_EL1**: Trace Buffer Status/syndrome Register

**TRBTRG_EL1**: Trace Buffer Trigger Counter Register

**TRACATR<n>**: Address Comparator Access Type Register <n>

**TRACVXR<n>**: Address Comparator Value Register <n>

**TRAUTHSTATUS**: Authentication Status Register

**TRCAUXCTL**: Auxiliary Control Register

**TRCBBCTLR**: Branch Broadcast Control Register

**TRCCCTL**: Cycle Count Control Register

**TRCCDCTRL0**: Context Identifier Comparator Control Register 0

**TRCCDCTRL1**: Context Identifier Comparator Control Register 1

**TRCCDCTR<n>**: Context Identifier Comparator Value Registers <n>

**TRCLAIMCLR**: Claim Tag Clear Register

**TRCLAIMSET**: Claim Tag Set Register

**TRCCNTCTR<n>**: Counter Control Register <n>

**TRCCNTRLDR<n>**: Counter Reload Value Register <n>
**TRC**

- **TRCCNTVR<n>**: Counter Value Register <n>
- **TRCCONFIGR**: Trace Configuration Register
- **TRCDEVARCH**: Device Architecture Register
- **TRCDEVII**: Device Configuration Register
- **TRCEVENTCTL0R**: Event Control 0 Register
- **TRCEVENTCTL1R**: Event Control 1 Register
- **TRCEXTINSELR<n>**: External Input Select Register <n>
- **TRCIDR0**: ID Register 0
- **TRCIDR1**: ID Register 1
- **TRCIDR10**: ID Register 10
- **TRCIDR11**: ID Register 11
- **TRCIDR12**: ID Register 12
- **TRCIDR13**: ID Register 13
- **TRCIDR2**: ID Register 2
- **TRCIDR3**: ID Register 3
- **TRCIDR4**: ID Register 4
- **TRCIDR5**: ID Register 5
- **TRCIDR6**: ID Register 6
- **TRCIDR7**: ID Register 7
- **TRCIDR8**: ID Register 8
- **TRCIDR9**: ID Register 9
- **TRCIMSPEC0**: IMP DEF Register 0
- **TRCIMSPEC<n>**: IMP DEF Register <n>
- **TRCOSLSR**: Trace OS Lock Status Register
- **TRCPRGCRLR**: Programming Control Register
- **TRCQCTL**: Q Element Control Register
- **TRCRSCTL<n>**: Resource Selection Control Register <n>
- **TRCRSR**: Resources Status Register
- **TRCSSEQVR<n>**: Sequencer State Transition Control Register <n>
- **TRCSERSTEV**: Sequencer Reset Control Register
- **TRCSSEQSTR**: Sequencer State Register
- **TRCSSCCR<n>**: Single-shot Comparator Control Register <n>
- **TRCSSCSR<n>**: Single-shot Comparator Control Status Register <n>
- **TRCSSPCICR<n>**: Single-shot Processing Element Comparator Input Control Register <n>
- **TRCCSTALLCTR**: Stall Control Register
- **TRCSTATR**: Trace Status Register
TRCSYNCPR: Synchronization Period Register
TRCTRACEIDR: Trace ID Register
TRCTSCTLR: Timestamp Control Register
TRCVICTL: ViewInst Main Control Register
TRCVIIECTLR: ViewInst Include/Exclude Control Register
TRCVIPCSSCTL: ViewInst Start/Stop PE Comparator Control Register
TRCVISSCTL: ViewInst Start/Stop Control Register
TRCVMIC: Virtual Context Identifier Comparator Control Register 0
TRCVMIC1: Virtual Context Identifier Comparator Control Register 1
TRCVI:<n>: Virtual Context Identifier Comparator Value Register <n>
TRFCR_EL1: Trace Filter Control Register (EL1)
TRFCR_EL2: Trace Filter Control Register (EL2)
TTBR0_EL1: Translation Table Base Register 0 (EL1)
TTBR0_EL2: Translation Table Base Register 0 (EL2)
TTBR0_EL3: Translation Table Base Register 0 (EL3)
TTBR1_EL1: Translation Table Base Register 1 (EL1)
TTBR1_EL2: Translation Table Base Register 1 (EL2)
UAO: User Access Override
VBAR_EL1: Vector Base Address Register (EL1)
VBAR_EL2: Vector Base Address Register (EL2)
VBAR_EL3: Vector Base Address Register (EL3)
VDISR_EL2: Virtual Deferred Interrupt Status Register
VMPIDR_EL2: Virtualization Multiprocessor ID Register
VNCR_EL2: Virtual Nested Control Register
VPIDR_EL2: Virtualization Processor ID Register
VSESRR_EL2: Virtual SError Exception Syndrome Register
VSTCR_EL2: Virtualization Secure Translation Control Register
VSTTBR_EL2: Virtualization Secure Translation Table Base Register
VTCR_EL2: Virtualization Translation Control Register
VTTBR_EL2: Virtualization Translation Table Base Register
ZCR_EL1: SVE Control Register for EL1
ZCR_EL2: SVE Control Register for EL2
ZCR_EL3: SVE Control Register for EL3
AArch64 System Instructions

AArch64 System Instructions

**AT S12E0R**: Address Translate Stages 1 and 2 EL0 Read
**AT S12E0W**: Address Translate Stages 1 and 2 EL0 Write
**AT S12E1R**: Address Translate Stages 1 and 2 EL1 Read
**AT S12E1W**: Address Translate Stages 1 and 2 EL1 Write
**AT S1E0R**: Address Translate Stage 1 EL0 Read
**AT S1E0W**: Address Translate Stage 1 EL0 Write
**AT S1E1R**: Address Translate Stage 1 EL1 Read
**AT S1E1WP**: Address Translate Stage 1 EL1 Write PAN
**AT S1E1RP**: Address Translate Stage 1 EL1 Read PAN
**AT S1E1RP**: Address Translate Stage 1 EL1 Write PAN
**AT S1E2R**: Address Translate Stage 1 EL2 Read
**AT S1E2W**: Address Translate Stage 1 EL2 Write
**AT S1E3R**: Address Translate Stage 1 EL3 Read
**AT S1E3W**: Address Translate Stage 1 EL3 Write

**CFP RCTX**: Control Flow Prediction Restriction by Context
**CPP RCTX**: Cache Prefetch Prediction Restriction by Context

**DC CGDSW**: Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way
**DC CGDVAC**: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC
**DC CGDVADP**: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoDP
**DC CGDVAP**: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP
**DC CGSW**: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by Set/Way
**DC CGVAC**: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC
**DC CGVADP**: Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoDP
**DC CGVAP**: Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP
**DC CIGDSW**: Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by Set/Way
**DC CIGDVAC**: Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by VA to PoC
**DC CIGSW**: Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by Set/Way
**DC CIGVAC**: Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by VA to PoC
**DC CISW**: Data or unified Cache line Clean and Invalidate by Set/Way
**DC CIVAC**: Data or unified Cache line Clean and Invalidate by VA to PoC
**DC CSW**: Data or unified Cache line Clean by Set/Way
**DC CVAC**: Data or unified Cache line Clean by VA to PoC
**DC CVADP**: Data or unified Cache line Clean by VA to PoDP
**DC CVAP**: Data or unified Cache line Clean by VA to PoP
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DC CVAU</strong></td>
<td>Data or unified Cache line Clean by VA to PoU</td>
</tr>
<tr>
<td><strong>DC GVA</strong></td>
<td>Data Cache set Allocation Tag by VA</td>
</tr>
<tr>
<td><strong>DC GZVA</strong></td>
<td>Data Cache set Allocation Tags and Zero by VA</td>
</tr>
<tr>
<td><strong>DC IGDSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way</td>
</tr>
<tr>
<td><strong>DC IGDVAC</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC</td>
</tr>
<tr>
<td><strong>DC IGSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Set/Way</td>
</tr>
<tr>
<td><strong>DC IGVAC</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC</td>
</tr>
<tr>
<td><strong>DC ISW</strong></td>
<td>Data or unified Cache line Invalidate by Set/Way</td>
</tr>
<tr>
<td><strong>DC IVAC</strong></td>
<td>Data or unified Cache line Invalidate by VA to PoC</td>
</tr>
<tr>
<td><strong>DC ZVA</strong></td>
<td>Data Cache Zero by VA</td>
</tr>
<tr>
<td><strong>DVP RCTX</strong></td>
<td>Data Value Prediction Restriction by Context</td>
</tr>
<tr>
<td><strong>IC IALLU</strong></td>
<td>Instruction Cache Invalidate All to PoU</td>
</tr>
<tr>
<td><strong>IC IALLUIS</strong></td>
<td>Instruction Cache Invalidate All to PoU, Inner Shareable</td>
</tr>
<tr>
<td><strong>IC IVAU</strong></td>
<td>Instruction Cache line Invalidate by VA to PoU</td>
</tr>
<tr>
<td><strong>S1_&lt;op1&gt;<em>&lt;Cn&gt;</em>&lt;Cm&gt;_&lt;op2&gt;</strong></td>
<td>IMPLEMENTATION DEFINED maintenance instructions</td>
</tr>
<tr>
<td><strong>TLBI ALLE1</strong></td>
<td>TLB Invalidate All, EL1</td>
</tr>
<tr>
<td><strong>TLBI ALLE1IS</strong></td>
<td>TLB Invalidate All, EL1, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI ALLE1OS</strong></td>
<td>TLB Invalidate All, EL1, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI ALLE2</strong></td>
<td>TLB Invalidate All, EL2</td>
</tr>
<tr>
<td><strong>TLBI ALLE2IS</strong></td>
<td>TLB Invalidate All, EL2, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI ALLE2OS</strong></td>
<td>TLB Invalidate All, EL2, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI ALLE3</strong></td>
<td>TLB Invalidate All, EL3</td>
</tr>
<tr>
<td><strong>TLBI ALLE3IS</strong></td>
<td>TLB Invalidate All, EL3, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI ALLE3OS</strong></td>
<td>TLB Invalidate All, EL3, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI ASIDE1</strong></td>
<td>TLB Invalidate by ASID, EL1</td>
</tr>
<tr>
<td><strong>TLBI ASIDE1IS</strong></td>
<td>TLB Invalidate by ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI ASIDE1OS</strong></td>
<td>TLB Invalidate by ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI ASIDE1OS</strong></td>
<td>TLB Invalidate by ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI IPAS2E1</strong></td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td><strong>TLBI IPAS2E1IS</strong></td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI IPAS2E1OS</strong></td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI IPAS2LE1</strong></td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td><strong>TLBI IPAS2LE1IS</strong></td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI IPAS2LE1OS</strong></td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td><strong>TLBI RIPAS2E1</strong></td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td><strong>TLBI RIPAS2E1IS</strong></td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td><strong>TLBI RIPAS2E1OS</strong></td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
</tbody>
</table>
TLBI RIPA$2LE1$: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

TLBI RIPA$2LE1$IS$: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

TLBI RIPA$2LE1$OS$: TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

TLBI RVAAE1$: TLB Range Invalidate by VA, All ASID, EL1

TLBI RVAAE1$IS$: TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

TLBI RVAAE1$OS$: TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

TLBI RVAALE1$: TLB Range Invalidate by VA, All ASID, Last level, EL1

TLBI RVAALE1$IS$: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBI RVAALE1$OS$: TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBI RVAE1$: TLB Range Invalidate by VA, EL1

TLBI RVAE1$IS$: TLB Range Invalidate by VA, EL1, Inner Shareable

TLBI RVAE1$OS$: TLB Range Invalidate by VA, EL1, Outer Shareable

TLBI RVAE2$: TLB Range Invalidate by VA, EL2

TLBI RVAE2$IS$: TLB Range Invalidate by VA, EL2, Inner Shareable

TLBI RVAE2$OS$: TLB Range Invalidate by VA, EL2, Outer Shareable

TLBI RVAE3$: TLB Range Invalidate by VA, EL3

TLBI RVAE3$IS$: TLB Range Invalidate by VA, EL3, Inner Shareable

TLBI RVAE3$OS$: TLB Range Invalidate by VA, EL3, Outer Shareable

TLBI RVALE1$: TLB Range Invalidate by VA, Last level, EL1

TLBI RVALE1$IS$: TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

TLBI RVALE1$OS$: TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

TLBI RVALE2$: TLB Range Invalidate by VA, Last level, EL2

TLBI RVALE2$IS$: TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

TLBI RVALE2$OS$: TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

TLBI RVALE3$: TLB Range Invalidate by VA, Last level, EL3

TLBI RVALE3$IS$: TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

TLBI RVALE3$OS$: TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

TLBI VAAE1$: TLB Invalidate by VA, All ASID, EL1

TLBI VAAE1$IS$: TLB Invalidate by VA, All ASID, EL1, Inner Shareable

TLBI VAAE1$OS$: TLB Invalidate by VA, All ASID, EL1, Outer Shareable

TLBI VAALE1$: TLB Invalidate by VA, All ASID, Last level, EL1

TLBI VAALE1$IS$: TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

TLBI VAALE1$OS$: TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

TLBI VAE1$: TLB Invalidate by VA, EL1

TLBI VAE1$IS$: TLB Invalidate by VA, EL1, Inner Shareable

TLBI VAE1$OS$: TLB Invalidate by VA, EL1, Outer Shareable
TLBI VAE2: TLB Invalidate by VA, EL2
TLBI VAE2IS: TLB Invalidate by VA, EL2, Inner Shareable
TLBI VAE2OS: TLB Invalidate by VA, EL2, Outer Shareable
TLBI VAE3: TLB Invalidate by VA, EL3
TLBI VAE3IS: TLB Invalidate by VA, EL3, Inner Shareable
TLBI VAE3OS: TLB Invalidate by VA, EL3, Outer Shareable
TLBI VALE1: TLB Invalidate by VA, Last level, EL1
TLBI VALE1IS: TLB Invalidate by VA, Last level, EL1, Inner Shareable
TLBI VALE1OS: TLB Invalidate by VA, Last level, EL1, Outer Shareable
TLBI VALE2: TLB Invalidate by VA, Last level, EL2
TLBI VALE2IS: TLB Invalidate by VA, Last level, EL2, Inner Shareable
TLBI VALE2OS: TLB Invalidate by VA, Last level, EL2, Outer Shareable
TLBI VALE3: TLB Invalidate by VA, Last level, EL3
TLBI VALE3IS: TLB Invalidate by VA, Last level, EL3, Inner Shareable
TLBI VALE3OS: TLB Invalidate by VA, Last level, EL3, Outer Shareable
TLBI VMALE1: TLB Invalidate by VMID, All at stage 1, EL1
TLBI VMALE1IS: TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable
TLBI VMALE1OS: TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable
TLBI VMALLS12E1: TLB Invalidate by VMID, All at Stage 1 and 2, EL1
TLBI VMALLS12E1IS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable
TLBI VMALLS12E1OS: TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable
The ACTLR_EL1 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

**Note**

Arm recommends the contents of this register have no effect on the PE when HCR_EL2.{E2H, TGE} is {1, 1}, and instead the configuration and control fields are provided by the ACTLR_EL2 register. This avoids the need for software to manage the contents of these register when switching between a Guest OS and a Host OS.

**Configuration**

AArch64 System register ACTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ACTLR[31:0].

AArch64 System register ACTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register ACTLR2[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ACTLR_EL1 is a 64-bit register.

**Field descriptions**

The ACTLR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | IMPLEMENTATION DEFINED |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | IMPLEMENTATION DEFINED |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**IMPLEMENTATION DEFINED, bits [63:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the ACTLR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ACTLR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    return NVMem[0x118];
  else
    return ACTLR_EL1;
elsif PSTATE.EL == EL2 then
  return ACTLR_EL1;
elsif PSTATE.EL == EL3 then
  return ACTLR_EL1;

MSR ACTLR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    NVMem[0x118] = X[t];
  else
    ACTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  ACTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  ACTLR_EL1 = X[t];
ACTLR_EL2, Auxiliary Control Register (EL2)

The ACTLR_EL2 characteristics are:

**Purpose**

Provides **IMPLEMENTATION DEFINED** configuration and control options for EL2.

---

**Note**

Arm recommends the contents of this register are updated to apply to EL0 when `HCR_EL2.{E2H, TGE}` is `{1, 1}`, gaining configuration and control fields from the `ACTLR_EL1`. This avoids the need for software to manage the contents of these registers when switching between a Guest OS and a Host OS.

---

**Configuration**

AArch64 System register ACTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register `HACTLR31:01`.

AArch64 System register ACTLR_EL2 bits [63:32] are architecturally mapped to AArch32 System register `HACTLR2[31:01]`.

If EL2 is not implemented, this register is `RES0` from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ACTLR_EL2 is a 64-bit register.

**Field descriptions**

The ACTLR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:0]**

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ACTLR_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, ACTLR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return ACTLR_EL2;
elsif PSTATE.EL == EL3 then
  return ACTLR_EL2;

MSR ACTLR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ACTLR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  ACTLR_EL2 = X[t];
ACTLR_EL3, Auxiliary Control Register (EL3)

The ACTLR_EL3 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED configuration and control options for EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to ACTLR_EL3 are UNDEFINED. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ACTLR_EL3 is a 64-bit register.

**Field descriptions**

The ACTLR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED. This field resets to an architecturally UNKNOWN value.

**Accessing the ACTLR_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, ACTLR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return ACTLR_EL3;

MSR ACTLR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  ACTLR_EL3 = X[t];
AFSR0_EL1, Auxiliary Fault Status Register 0 (EL1)

The AFSR0_EL1 characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

**Configuration**

AArch64 System register AFSR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ADFSR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AFSR0_EL1 is a 64-bit register.

**Field descriptions**

The AFSR0_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the AFSR0_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic AFSR0_EL1 or AFSR0_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, AFSR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGTR_EL2.AFSR0_EL1 == '1' then
        return NVMem[0x128];
    else
        return AFSR0_EL1;
    end if;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR0_EL2;
    else
        return AFSR0_EL1;
    end if;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL1;
end if;

MSR AFSR0_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.AFSR0_EL1 == '1' then
        AFSR0_EL1 = X[t];
    else
        AFSR0_EL1 = X[t];
    end if;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR0_EL2 = X[t];
    else
        AFSR0_EL1 = X[t];
    end if;
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t];
end if;

MRS <Xt>, AFSR0_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x128];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return AFSR0_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return AFSR0_EL1;
  else
    UNDEFINED;

MSR AFSR0_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b00</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x128] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    AFSR0_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    AFSR0_EL1 = X[t];
  else
    UNDEFINED;
AFSR0_EL2, Auxiliary Fault Status Register 0 (EL2)

The AFSR0_EL2 characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL2.

**Configuration**

AArch64 System register AFSR0_EL2 bits [31:0] are architecturally mapped to AArch32 System register HADFSR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AFSR0_EL2 is a 64-bit register.

**Field descriptions**

The AFSR0_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the AFSR0_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic AFSR0_EL2 or AFSR0_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, AFSR0_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return AFSR0_EL2;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL2;

MSR AFSR0_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.AFSR0_EL1 == '1' then
        return NVMem[0x128];
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR0_EL2;
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL1;

MSR AFSR0_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

MRS <Xt>, AFSR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAAarch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAarch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.AFSR0_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAarch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x128] = X[t];
    else
        AFSR0_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR0_EL2 = X[t];
    else
        AFSR0_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t];
AFSR0_EL3, Auxiliary Fault Status Register 0 (EL3)

The AFSR0_EL3 characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to AFSR0_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AFSR0_EL3 is a 64-bit register.

**Field descriptions**

The AFSR0_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:0]**

This field resets to an architecturally UNKNOWN value.

**Accessing the AFSR0_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, AFSR0_EL3

```plaintext
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  UNDEFINED;
elif PSTATE.EL == EL2 then
  UNDEFINED;
elif PSTATE.EL == EL3 then
  return AFSR0_EL3;

MSR AFSR0_EL3, <Xt>

```plaintext
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>
```

Page 30
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AFSR0_EL3 = X[t];
AFSR1_EL1, Auxiliary Fault Status Register 1 (EL1)

The AFSR1_EL1 characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

**Configuration**

AArch64 System register AFSR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register AIFSR[31:0]. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AFSR1_EL1 is a 64-bit register.

**Field descriptions**

The AFSR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>bit</th>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:0]**

This field resets to an architecturally UNKNOWN value.

**Accessing the AFSR1_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic AFSR1_EL1 or AFSR1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, AFSR1_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.AFSR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x130];
    else
        return AFSR1_EL1;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            return AFSR1_EL2;
        else
            return AFSR1_EL1;
        elsif PSTATE.EL == EL3 then
            return AFSR1_EL1;
    MSR AFSR1_EL1, <Xt>

    op0  op1  CRn  CRm  op2
    0b11  0b000  0b0101  0b0001  0b001

    if PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGWTR_EL2.AFSR1_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x130] = X[t];
        else
            AFSR1_EL1 = X[t];
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.E2H == '1' then
                AFSR1_EL2 = X[t];
            else
                AFSR1_EL1 = X[t];
        elsif PSTATE.EL == EL3 then
            AFSR1_EL1 = X[t];
        MRS <Xt>, AFSR1_EL12

    op0  op1  CRn  CRm  op2
    0b11  0b101  0b0101  0b0001  0b001

    if PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGRTR_EL2.AFSR1_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x130] = X[t];
        else
            AFSR1_EL1 = X[t];
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.E2H == '1' then
                AFSR1_EL2 = X[t];
            else
                AFSR1_EL1 = X[t];
        elsif PSTATE.EL == EL3 then
            AFSR1_EL1 = X[t];
        MRS <Xt>, AFSR1_EL12

    op0  op1  CRn  CRm  op2
    0b11  0b101  0b0101  0b0001  0b001

Page 33
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() \&\& HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x130];
  elsif EL2Enabled() \&\& HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() \&\& HCR_EL2.E2H == '1' then
    return AFSR1_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() \&\& HCR_EL2.E2H == '1' then
    return AFSR1_EL1;
  else
    UNDEFINED;

MSR AFSR1_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() \&\& HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x130] = X[t];
  elsif EL2Enabled() \&\& HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() \&\& HCR_EL2.E2H == '1' then
    AFSR1_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() \&\& HCR_EL2.E2H == '1' then
    AFSR1_EL1 = X[t];
  else
    UNDEFINED;
AFSR1_EL2, Auxiliary Fault Status Register 1 (EL2)

The AFSR1_EL2 characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL2.

**Configuration**

AArch64 System register AFSR1_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAIFSR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AFSR1_EL2 is a 64-bit register.

**Field descriptions**

The AFSR1_EL2 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ![IMPLEMENTATION DEFINED](image1) | ![IMPLEMENTATION DEFINED](image2) | ![IMPLEMENTATION DEFINED](image3) | ![IMPLEMENTATION DEFINED](image4) | ![IMPLEMENTATION DEFINED](image5) | ![IMPLEMENTATION DEFINED](image6) | ![IMPLEMENTATION DEFINED](image7) | ![IMPLEMENTATION DEFINED](image8) | ![IMPLEMENTATION DEFINED](image9) | ![IMPLEMENTATION DEFINED](image10) | ![IMPLEMENTATION DEFINED](image11) | ![IMPLEMENTATION DEFINED](image12) | ![IMPLEMENTATION DEFINED](image13) | ![IMPLEMENTATION DEFINED](image14) | ![IMPLEMENTATION DEFINED](image15) | ![IMPLEMENTATION DEFINED](image16) | ![IMPLEMENTATION DEFINED](image17) | ![IMPLEMENTATION DEFINED](image18) | ![IMPLEMENTATION DEFINED](image19) | ![IMPLEMENTATION DEFINED](image20) | ![IMPLEMENTATION DEFINED](image21) | ![IMPLEMENTATION DEFINED](image22) | ![IMPLEMENTATION DEFINED](image23) | ![IMPLEMENTATION DEFINED](image24) | ![IMPLEMENTATION DEFINED](image25) | ![IMPLEMENTATION DEFINED](image26) | ![IMPLEMENTATION DEFINED](image27) | ![IMPLEMENTATION DEFINED](image28) | ![IMPLEMENTATION DEFINED](image29) | ![IMPLEMENTATION DEFINED](image30) | ![IMPLEMENTATION DEFINED](image31) | ![IMPLEMENTATION DEFINED](image32) |

**IMPLEMENTATION DEFINED, bits [63:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the AFSR1_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic AFSR1_EL2 or AFSR1_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return AFSR1_EL2;
elsif PSTATE.EL == EL3 then
  return AFSR1_EL2;

MSR AFSR1_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  AFSR1_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  AFSR1_EL2 = X[t];

MRS <Xt>, AFSR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRTR_EL2.AFSR1_EL1 == '1' then
    return NVMem[0x130];
  else
    return AFSR1_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return AFSR1_EL2;
  else
    return AFSR1_EL1;
elsif PSTATE.EL == EL3 then
  return AFSR1_EL1;

MSR AFSR1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.AFSR1_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x130] = X[t];
    else
        AFSR1_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR1_EL2 = X[t];
    else
        AFSR1_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    AFSR1_EL1 = X[t];
end
AFSR1_EL3, Auxiliary Fault Status Register 1 (EL3)

The AFSR1_EL3 characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to AFSR1_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AFSR1_EL3 is a 64-bit register.

**Field descriptions**

The AFSR1_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMPLEMENTATION DEFINED, bits [63:0]

This field resets to an architecturally UNKNOWN value.

**Accessing the AFSR1_EL3**

Accesses to this register use the following encodings:

**MRS <Xt>, AFSR1_EL3**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return AFSR1_EL3;

**MSR AFSR1_EL3, <Xt>**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AFSR1_EL3 = X[t];
AIDR_EL1, Auxiliary ID Register

The AIDR_EL1 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED identification information.

The value of this register must be interpreted in conjunction with the value of MIDR_EL1.

**Configuration**

AArch64 System register AIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register AIDR[31:0].

**Attributes**

AIDR_EL1 is a 64-bit register.

**Field descriptions**

The AIDR_EL1 bit assignments are:

```
 63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32
IMPLMNTATION DEFINED
IMPLMNTATION DEFINED
 31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  09  08  07  06  05  04  03  02  01  00
```

**IMPLEMENTATION DEFINED**, bits [63:0]

**IMPLEMENTATION DEFINED.**

**Accessing the AIDR_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, AIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b11</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRTR_EL2.AIDR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return AIDR_EL1;
eelsif PSTATE.EL == EL2 then
  return AIDR_EL1;
eelsif PSTATE.EL == EL3 then
  return AIDR_EL1;
AMAIR_EL1, Auxiliary Memory Attribute Indirection Register (EL1)

The AMAIR_EL1 characteristics are:

**Purpose**

Provides implementation defined memory attributes for the memory regions specified by MAIR_EL1.

**Configuration**

AArch64 System register AMAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register AMAIR0[31:0].

AArch64 System register AMAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register AMAIR1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMAIR_EL1 is a 64-bit register.

**Field descriptions**

The AMAIR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

AMAIR_EL1 is permitted to be cached in a TLB.

**IMPLEMENTATION DEFINED, bits [63:0]**

This field resets to an architecturally UNKNOWN value.

**Accessing the AMAIR_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic AMAIR_EL1 or AMAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, AMAIR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.AMAIR_EL1 == '1' then
    return AMAIR_EL1;
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  end
elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.AMAIR_EL1 == '111' then
  return NVMem[0x148];
else
  return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return AMAIR_EL2;
  else
    return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
  return AMAIR_EL1;

MSR AMAIR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.AMAIR_EL1 == '1' then
    return AMAIR_EL1;
  else
    AMAIR_EL1 = X[t];
  end
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    AMAIR_EL2 = X[t];
  else
    AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  AMAIR_EL1 = X[t];

MRS <Xt>, AMAIR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x148];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return AMAIR_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return AMAIR_EL1;
  else
    UNDEFINED;

MSR AMAIR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x148] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    AMAIR_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    AMAIR_EL1 = X[t];
  else
    UNDEFINED;
AMAIR_EL2, Auxiliary Memory Attribute Indirection Register (EL2)

The AMAIR_EL2 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL2.

**Configuration**

AArch64 System register AMAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HAMAIR0[31:0].

AArch64 System register AMAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HAMAIR1[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMAIR_EL2 is a 64-bit register.

**Field descriptions**

The AMAIR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

AMAIR_EL2 is permitted to be cached in a TLB.

**Accessing the AMAIR_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic AMAIR_EL2 or AMAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

```
MRS <Xt>, AMAIR_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return AMAIR_EL2;
elsif PSTATE.EL == EL3 then
  return AMAIR_EL2;

MSR AMAIR_EL2, <Xt>

\[
\begin{array}{cccc}
  \text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
  0b11 & 0b100 & 0b1010 & 0b0011 & 0b000 \\
\end{array}
\]

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.AMAIR_EL1 == '1' then
    return NVMem[0x148];
  else
    return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return AMAIR_EL2;
  else
    return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
  return AMAIR_EL1;

MSR AMAIR_EL1, <Xt>

\[
\begin{array}{cccc}
  \text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
  0b11 & 0b000 & 0b1010 & 0b0011 & 0b000 \\
\end{array}
\]

MRS <Xt>, AMAIR_EL1

\[
\begin{array}{cccc}
  \text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
  0b11 & 0b000 & 0b1010 & 0b0011 & 0b000 \\
\end{array}
\]

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && (!ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x148];
  else
    return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return AMAIR_EL2;
  else
    return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
  return AMAIR_EL1;

MSR AMAIR_EL1, <Xt>

\[
\begin{array}{cccc}
  \text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
  0b11 & 0b000 & 0b1010 & 0b0011 & 0b000 \\
\end{array}
\]

AMAIR_EL2, Auxiliary Memory Attribute Indirection Register (EL2)
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.AMAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x148] = X[t];
    else
        AMAIR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AMAIR_EL2 = X[t];
    else
        AMAIR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t];
else
    AMAIR_EL1 = X[t];

AMAIR_EL3, Auxiliary Memory Attribute Indirection Register (EL3)

The AMAIR_EL3 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR_EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to AMAIR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMAIR_EL3 is a 64-bit register.

**Field descriptions**

The AMAIR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | IMPLEMENTATION DEFINED |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

AMAIR_EL3 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the AMAIR_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, AMAIR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsf PSTATE.EL == EL1 then
    UNDEFINED;
elsf PSTATE.EL == EL2 then
    UNDEFINED;
elsf PSTATE.EL == EL3 then
    return AMAIR_EL3;
### MSR AMAIR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AMAIR_EL3 = X[t];
```

---

27/09/2019 18:22; 6134483bd14d8c12a99c984cb6e3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The AMCFGR_EL0 characteristics are:

**Purpose**

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR_EL0 is applicable to both the architected and the auxiliary counter groups.

**Configuration**

AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCFGR[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCFGR_EL0 are UNDEFINED.

**Attributes**

AMCFGR_EL0 is a 64-bit register.

**Field descriptions**

The AMCFGR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:32]</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>[31:28]</td>
<td>NCG</td>
<td>Defines the number of counter groups.</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**NCG, bits [31:28]**

Defines the number of counter groups.

The number of implemented counter groups is defined as \([\text{AMCFGR_EL0.NCG} + 1]\).

If the number of implemented auxiliary activity monitor event counters is zero, this field has a value of 0b0000. Otherwise, this field has a value of 0b0001.

**Bits [27:25]**

Reserved, RES0.

**HDBG, bit [24]**

Halt-on-debug supported.

From Armv8, this feature must be supported, and so this bit is 0b1.
<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AMCR_EL0.HDBG is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>AMCR_EL0.HDBG is read/write.</td>
</tr>
</tbody>
</table>

**Bits [23:14]**

Reserved, `RAZ`.

**SIZE, bits [13:8]**

Defines the size of activity monitor event counters.

The size of the activity monitor event counters implemented by the activity monitors Extension is defined as \([\text{AMCFGR_EL0.SIZE} + 1]\).

From Armv8, the counters are 64-bit, and so this field is `0b111111`.

---

**Note**

Software also uses this field to determine the spacing of counters in the memory-map. From Armv8, the counters are at doubleword-aligned addresses.

---

**N, bits [7:0]**

Defines the number of activity monitor event counters.

The total number of counters implemented in all groups by the Activity Monitors Extension is defined as \([\text{AMCFGR_EL0.N} + 1]\).

**Accessing the AMCFGR_EL0**

Accesses to this register use the following encodings:

\[
\text{MRS <Xt>, AMCFGR_EL0}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCFGR_EL0;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCFGR_EL0;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCFGR_EL0;
    end
elsif PSTATE.EL == EL3 then
    return AMCFGR_EL0;
AMCG1IDR_EL0, Activity Monitors Counter Group 1 Identification Register

The AMCG1IDR_EL0 characteristics are:

**Purpose**

Defines which auxiliary counters are implemented, and which of them have a corresponding virtual offset register, AMEVCNTVOFF1<\n>EL2 implemented.

**Configuration**

This register is present only when ARMv8.6-AMU is implemented. Otherwise, direct accesses to AMCG1IDR_EL0 are UNDEFINED.

**Attributes**

AMCG1IDR_EL0 is a 64-bit register.

**Field descriptions**

The AMCG1IDR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>AMEVCNTOFF1&lt;\n&gt;EL2, bit [n+16], for n = 0 to 15</td>
</tr>
<tr>
<td>61</td>
<td>AMEVCNTR1&lt;\n&gt;EL0, bit [n], for n = 0 to 15</td>
</tr>
<tr>
<td>31</td>
<td>Bits [63:32]</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**AMEVCNTOFF1<\n>EL2, bit [n+16], for n = 0 to 15**

Indicates which implemented auxiliary counters have a corresponding virtual offset register, AMEVCNTVOFF1<\n>EL2 implemented.

<table>
<thead>
<tr>
<th>AMEVCNTOFF1&lt;\n&gt;EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, mean that AMEVCNTR1&lt;\n&gt;EL0 does not have an offset, or is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the offset AMEVCNTVOFF1&lt;\n&gt;EL2 is implemented for AMEVCNTR1&lt;\n&gt;EL0.</td>
</tr>
</tbody>
</table>

**AMEVCNTR1<\n>EL0, bit [n], for n = 0 to 15**

Indicates which auxiliary counters AMEVCNTR1<\n>EL0 are implemented.

<table>
<thead>
<tr>
<th>AMEVCNTR1&lt;\n&gt;EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR1&lt;\n&gt;EL0 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR1&lt;\n&gt;EL0 is implemented.</td>
</tr>
</tbody>
</table>
Accessing the AMCG1IDR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, AMCG1IDR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64SystemAccessTrap(EL3, 0x18);
    else
        return AMCG1IDR_EL0;
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCG1IDR_EL0;
        endif
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCG1IDR_EL0;
        endif
    elsif PSTATE.EL == EL3 then
        return AMCG1IDR_EL0;
    endif

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The AMCGCR_EL0 characteristics are:

**Purpose**

Provides information on the number of activity monitor event counters implemented within each counter group.

**Configuration**

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCGCR[31:0].

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to External register AMCGCR[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCGCR_EL0 are UNDEFINED.

**Attributes**

AMCGCR_EL0 is a 64-bit register.

**Field descriptions**

The AMCGCR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | CG1NC | CG0NC |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:16]**

Reserved, RES0.

**CG1NC, bits [15:8]**

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In AMUv1, the permitted range of values is 0x0 to 0x10.

**CG0NC, bits [7:0]**

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

In AMUv1, the value of this field is 0x4.

**Accessing the AMCGCR_EL0**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCGCR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCGCR_EL0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCGCR_EL0;
elsif PSTATE.EL == EL3 then
    return AMCGCR_EL0;

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

MRS <Xt>, AMCGCR_EL0
AMCNTENCLR0_EL0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0_EL0 characteristics are:

**Purpose**

Disable control bits for the architected activity monitors event counters, AMEVCTR0<n>_EL0.

**Configuration**

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR0[31:0].

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENCLR0_EL0 is a 64-bit register.

**Field descriptions**

The AMCNTENCLR0_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | P<n>, bit [n] |

**Bits [63:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter disable bit for AMEVCTR0<n>_EL0.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR_EL0.CG0NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCTR0&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCTR0&lt;n&gt;_EL0 is enabled. When written, disables AMEVCTR0&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
Accessing the AMCNTENCLR0_EL0

Accesses to this register use the following encodings:

MRS <Xt>, AMCNTENCLR0_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && !HaveEL(EL3) ||
                SCR_EL3.FGTEn == '1' && !HaveEL(EL3) ||
                HAFGRTR_EL2.AMCNTEN0 == '1' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return AMCNTENCLR0_EL0;
        endif
    else
        return AMCNTENCLR0_EL0;
    endif
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
        SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCNTENCLR0_EL0;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCNTENCLR0_EL0;
    endif
elsif PSTATE.EL == EL3 then
    return AMCNTENCLR0_EL0;

MSR AMCNTENCLR0_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if IsHighestEL(PSTATE.EL) then
    AMCNTENCLR0_EL0 = <t>;
else
    UNDEFINED;

27/09/2019 18:23, 6134483bd14dc8c12a99c984cb6e3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
AMCNTENCLR1_EL0, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1_EL0 characteristics are:

**Purpose**

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>_EL0.

**Configuration**

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR1[31:0].

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENCLR1_EL0 is a 64-bit register.

**Field descriptions**

The AMCNTENCLR1_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RESO|

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----| RESO|

<table>
<thead>
<tr>
<th>P&lt;n&gt;, bit [n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter disable bit for AMEVCNTR1<n>_EL0.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR_EL0.CG1NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR1&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR1&lt;n&gt;_EL0 is enabled. When written, disables AMEVCNTR1&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
Accessing the AMCNTENCLR1_EL0

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENCLR1_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

**Note**

The number of auxiliary activity monitor event counters implemented is zero exactly when AMCFGR_EL0.NCG == 0b0000.

Accesses to this register use the following encodings:

MRS <Xt>, AMCNTENCLR1_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCNTENCLR1_EL0;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && !HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCNTENCLR1_EL0;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCNTENCLR1_EL0;
    end
elsif PSTATE.EL == EL3 then
    return AMCNTENCLR1_EL0;

MSR AMCNTENCLR1_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if IsHighestEL(PSTATE.EL) then
  AMCNTENCLR1_EL0 = X[t];
else
  UNDEFINED;
AMCNTENSET0_EL0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0_EL0 characteristics are:

**Purpose**

Enable control bits for the architected activity monitors event counters, AMEVCNTR0<n>_EL0.

**Configuration**

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET0[31:0].

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET0[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENSET0_EL0 is a 64-bit register.

**Field descriptions**

The AMCNTENSET0_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>61</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter enable bit for AMEVCNTR0<n>_EL0.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR_EL0.CG0NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR0&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR0&lt;n&gt;_EL0 is enabled. When written, enables AMEVCNTR0&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
Accessing the AMCNTENSET0_EL0

Accesses to this register use the following encodings:

**MRS <Xt>, AMCNTENSET0_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return AMCNTENSET0_EL0;
  end
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return AMCNTENSET0_EL0;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMCNTENSET0_EL0;
  end
elsif PSTATE.EL == EL3 then
  return AMCNTENSET0_EL0;

**MSR AMCNTENSET0_EL0, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if IsHighestEL(PSTATE.EL) then
  AMCNTENSET0_EL0 = X[t];
else
  UNDEFINED;

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
AMCNTENSET1_EL0, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1_EL0 characteristics are:

**Purpose**

Enable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>_EL0.

**Configuration**

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET1[31:0].

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENSET1_EL0 is a 64-bit register.

**Field descriptions**

The AMCNTENSET1_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | P<n>, bit [n] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter enable bit for AMEVCNTR1<n>_EL0.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR_EL0.CG1NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR1&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR1&lt;n&gt;_EL0 is enabled. When written, enables AMEVCNTR1&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
Accessing the AMCNTENSET1_EL0

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENSET1_EL0 are constrained unpredictable, and the following behaviors are permitted:

- Accesses to the register are undefined.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

**Note**

The number of auxiliary activity monitor counters implemented is zero when AMCFGR_EL0.NCG == 0b0000.

Accesses to this register use the following encodings:

**MRS <Xt>, AMCNTENSET1_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '1' &&
    HAFGRTR_EL2.AMCNTEN1 == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMCNTENSET1_EL0;
else
  if !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif! ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') &&
    HAFGRTR_EL2.AMCNTEN1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMCNTENSET1_EL0;
endif
```

**MSR AMCNTENSET1_EL0, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if IsHighestEL(PSTATE.EL) then
    AMCNTENSET1_EL0 = X[t];
else
    UNDEFINED;
AMCR_EL0, Activity Monitors Control Register

The AMCR_EL0 characteristics are:

**Purpose**

Global control register for the activity monitors implementation. AMCR_EL0 is applicable to both the architected and the auxiliary counter groups.

**Configuration**

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMCR[31:0].

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to External register AMCR[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCR_EL0 are UNDEFINED.

**Attributes**

AMCR_EL0 is a 64-bit register.

**Field descriptions**

The AMCR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>CG1RZ</td>
</tr>
<tr>
<td>16-10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9-5</td>
<td>HDBG</td>
</tr>
<tr>
<td>4-0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:18]**

Reserved, RES0.

**CG1RZ, bit [17]**

When ARMv8.6-AMU is implemented:

Counter Group 1 Read Zero.

<table>
<thead>
<tr>
<th>CG1RZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>System register reads of AMEVCNTR1&lt;n&gt;_EL0 return the event count at all implemented and enabled Exception levels.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the current Exception level is the highest implemented Exception level, system register reads of AMEVCNTR1&lt;n&gt;_EL0 return the event count. Otherwise, reads of AMEVCNTR1&lt;n&gt;_EL0 return a zero value.</td>
</tr>
</tbody>
</table>

**Note**

Reads from the memory-mapped view are unaffected by this field.

**Otherwise:**

Reserved, RES0.
Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Activity monitors do not halt counting when the PE is halted in Debug state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Activity monitors halt counting when the PE is halted in Debug state.</td>
</tr>
</tbody>
</table>

Bits [9:0]

Reserved, RES0.

Accessing the AMCR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, AMCR_EL0

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCR_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCR_EL0;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCR_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCR_EL0;

MSR AMCR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b11</td>
<td>0b11</td>
<td>0b11</td>
</tr>
</tbody>
</table>
if IsHighestEL(PSTATE.EL) then
    AMCR_EL0 = \( X[t] \);
else
    UNDEFINED;
**AMEVCNTR0<n>_EL0, Activity Monitors Event Counter Registers 0, n = 0 - 15**

The AMEVCNTR0<n>_EL0 characteristics are:

**Purpose**

Provides access to the architected activity monitor event counters.

**Configuration**

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to AArch32 System register AMEVCNTR0<n>[63:0].

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR0<n>[63:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n>_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMEVCNTR0<n>_EL0 is a 64-bit register.

**Field descriptions**

The AMEVCNTR0<n>_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**ACNT, bits [63:0]**

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If ARMv8.6-AMU is implemented, HCR_EL2.AMVOFFEN is 1, SCR_EL3.AMVOFFEN is 0, HCR_EL2.{E2H, TGE} is not {1,1}, and EL2 is implemented in the current Security state, access to these registers at EL0 or EL1 return (PCount<63:0> - AMEVCNTRVOFF0<n>_EL2<63:0>).

PCount is the physical count returned when AMEVCNTR0<n>_EL0 is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

**Accessing the AMEVCNTR0<n>_EL0**

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:
• Accesses to the register are UNDEFINED.
• Accesses to the register behave as RAZ/WI.
• Accesses to the register execute as a NOP.

Note

AMCGCR_EL0.CG0NC identifies the number of architectured activity monitor event counters.

Accesses to this register use the following encodings:

MRS <Xt>, AMEVCNTR0<n>_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b010:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR0<n>_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
  end
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') &&
    HAFGRTR_EL2.AMEVCNTR0<n>_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
  end
elsif PSTATE.EL == EL3 then
  return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
end

MSR AMEVCNTR0<n>_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b010:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if IsHighestEL(PSTATE.EL) then
  AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
  UNDEFINED;

27/09/2019 18:23; 6134483bd14dc8c12a99c984cb6e3431cc1c9707
Page 71
The AMEVCNTR1<n>_EL0 characteristics are:

**Purpose**

Provides access to the auxiliary activity monitor event counters.

**Configuration**

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to AArch32 System register AMEVCNTR1<n>[63:0].

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to External register AMEVCNTR1<n>[63:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n>_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMEVCNTR1<n>_EL0 is a 64-bit register.

**Field descriptions**

The AMEVCNTR1<n>_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | ACNT |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | ACNT |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**ACNT, bits [63:0]**

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If ARMv8.6-AMU is implemented, HCR_EL2.AMVOFFEN is 1, SCR_EL3.AMVOFFEN is 0, HCR_EL2 {E2H, TGE} is not {1,1}, EL2 is implemented in the current Security state, and AMCR_EL0 CG1RZ is 0, reads to these registers at EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF1<n>_EL2<63:0>).

PCount is the physical count returned when AMEVCNTR1<n>_EL0 is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

**Accessing the AMEVCNTR1<n>_EL0**

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:
• Accesses to the register are **UNDEFINED**.
• Accesses to the register behave as **RAZ/WI**.
• Accesses to the register execute as a **NOP**.

**Note**

**AMCGCR_EL0.CG1NC** identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

**MRS <Xt>, AMEVCNTR1<n>_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b110:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end
    else
        !ELUsingAArch32(EL1) && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        !ELUsingAArch32(EL2) && SCRO_EL3.FGTEn == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
            Zeros();
    end
    return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            !ELUsingAArch32(EL2) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            end
        else
            !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
                Zeros();
        end
        return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
                Zeros();
        end
        return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
    else
        if PSTATE.EL == EL3 then
            return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
        end
    end
end
```

**MSR AMEVCNTR1<n>_EL0, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b110:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if IsHighestEL(PSTATE.EL) then
    AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;

AMEVCNTVOFF0\(<n>_EL2\), Activity Monitors Event Counter Virtual Offset Registers 0, \(n = 0\) - 15

The AMEVCNTVOFF0\(<n>_EL2\) characteristics are:

**Purpose**

Holds the 64-bit virtual offset for architected activity monitor events.

**Configuration**

This register is present only when ARMv8.6-AMU is implemented. Otherwise, direct accesses to AMEVCNTVOFF0\(<n>_EL2\) are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

AMEVCNTVOFF0\(<n>_EL2\) is a 64-bit register.

**Field descriptions**

The AMEVCNTVOFF0\(<n>_EL2\) bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual offset</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual offset.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the AMEVCNTVOFF0\(<n>_EL2**

If \(<n>\) is not 0, 2 or 3, reads and writes of AMEVCNTVOFF0\(<n>_EL2\) are **UNDEFINED**.

Accesses to this register use the following encodings:

\[
\begin{array}{c|c|c|c|c}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b11 & 0b100 & 0b1101 & 0b100:n[3] & n[2:0] \\
\end{array}
\]
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0xA00+8*UInt(CRm<0>:op2<2:0>)];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.AMVOFFEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMEVCNTVOFF0_EL2[UInt(CRm<0>:op2<2:0>)];
  end if;
elsif PSTATE.EL == EL3 then
  return AMEVCNTVOFF0_EL2[UInt(CRm<0>:op2<2:0>)];
end if;

MSR AMEVCNTVOFF0<n>_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b100:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0xA00+8*UInt(CRm<0>:op2<2:0>)] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  end if;
elsif PSTATE.EL == EL2 then
  AMEVCNTVOFF0_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
  AMEVCNTVOFF0_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];

The AMEVCNTVOFF1<n>_EL2 characteristics are:

**Purpose**

Holds the 64-bit virtual offset for auxiliary activity monitor events.

**Configuration**

This register is present only when ARMv8.6-AMU is implemented. Otherwise, direct accesses to AMEVCNTVOFF1<n>_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMEVCNTVOFF1<n>_EL2 is a 64-bit register.

**Field descriptions**

The AMEVCNTVOFF1<n>_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:0]**

Virtual offset.

This field resets to an architecturally UNKNOWN value.

**Accessing the AMEVCNTVOFF1<n>_EL2**

**Note**

AMECG1IDR_EL0 identifies which auxiliary activity monitor event counters have a corresponding virtual offset implemented.

Accesses to this register use the following encodings:

MRS <Xt>, AMEVCNTVOFF1<n>_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b101:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0xA80+8*UInt(CRm<0>:op2<2:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.AMVOFFEN == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTVOFF1_EL2[UInt(CRm<0>:op2<2:0>)];
    end
elsif PSTATE.EL == EL3 then
    return AMEVCNTVOFF1_EL2[UInt(CRm<0>:op2<2:0>)];

MSR AMEVCNTVOFF1<n>_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b101:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0xA80+8*UInt(CRm<0>:op2<2:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    AMEVCNTVOFF1_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    AMEVCNTVOFF1_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
AMEVTYPER0<n>_EL0, Activity Monitors Event Type Registers 0, n = 0 - 15

The AMEVTYPER0<n>_EL0 characteristics are:

**Purpose**

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n>_EL0 counts.

**Configuration**

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0].

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER0<n>[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n>_EL0 are UNDEFINED.

**Attributes**

AMEVTYPER0<n>_EL0 is a 64-bit register.

**Field descriptions**

The AMEVTYPER0<n>_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>61</td>
<td>evtCount, bits [15:0]</td>
</tr>
<tr>
<td>60</td>
<td>Event to count. The event number of the event</td>
</tr>
<tr>
<td></td>
<td>that is counted by the architected activity</td>
</tr>
<tr>
<td></td>
<td>monitor event counter AMEVCNTR0&lt;n&gt;_EL0. The</td>
</tr>
<tr>
<td></td>
<td>value of this field is architecturally</td>
</tr>
<tr>
<td></td>
<td>mandated for each architected counter.</td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>_EL0. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

<table>
<thead>
<tr>
<th>evtCount</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x8011</td>
<td>Processor frequency cycles</td>
<td>When n == 0</td>
</tr>
<tr>
<td>0x4004</td>
<td>Constant frequency cycles</td>
<td>When n == 1</td>
</tr>
<tr>
<td>0x8008</td>
<td>Instructions retired</td>
<td>When n == 2</td>
</tr>
<tr>
<td>0x4005</td>
<td>Memory stall cycles</td>
<td>When n == 3</td>
</tr>
</tbody>
</table>

**Accessing the AMEVTYPER0<n>_EL0**

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n>_EL0 are constrained unpredictable, and the following behaviors are permitted:

- Accesses to the register are undefined.
- Accesses to the register behave as RAZ/WI.
Accesses to the register execute as a NOP.

**Note**

`AMCGCR_EL0.CG0NC` identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

**MRS <Xt>, AMEVTYPER0<n>_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b1101</td>
<td>0b011:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
  elsif PSTATE.EL == EL3 then
    return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
```

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
AMEVTYPER1<n>_EL0, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n>_EL0 characteristics are:

**Purpose**

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1<n>_EL0 counts.

**Configuration**

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER1<n>[31:0].

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to External register AMEVTYPER1<n>[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n>_EL0 are UNDEFINED.

**Attributes**

AMEVTYPER1<n>_EL0 is a 64-bit register.

**Field descriptions**

The AMEVTYPER1<n>_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

Bits [63:16]

Reserved, RES0.

**evtCount, bits [15:0]**

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1<n>_EL0.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1<n>_EL0, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCNTR1<n>_EL0 might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCNTR1<n>_EL0 is enabled, writes to this register have UNPREDICTABLE results.
Accessing the AMEVTYPER1\(<n>_EL0\) Activity Monitors Event Type Registers 1, \(n = 0 \cdot 15\)

If \(<n>_\) is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1\(<n>_EL0\) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

**Note**

AMEGCR_EL0.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

**MRS \(<Xt>, AMEVTYPER1\(<n>_EL0\)**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm[3]</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b111 : n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
  end
endif

elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVTYPER1\(<n>_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
  end
endif

elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
  end
endif

elsif PSTATE.EL == EL3 then
  return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
endif

**MSR AMEVTYPER1\(<n>_EL0, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm[3]</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b111 : n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if IsHighestEL(PSTATE.EL) then
  AMEVTYPE1_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
  UNDEFINED;
The AMUSERENR_EL0 characteristics are:

**Purpose**

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR_EL0 is applicable to both the architected and the auxiliary counter groups.

**Configuration**

AArch64 System register AMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register AMUSERENR[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR_EL0 are UNDEFINED.

**Attributes**

AMUSERENR_EL0 is a 64-bit register.

**Field descriptions**

The AMUSERENR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | RES0 | EN |

**Bits [63:1]**

Reserved, RES0.

**EN, bit [0]**

Traps EL0 accesses to the activity monitors registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped, reported using EC syndrome value 0x18:
  - AMCFGR_ELO, AMCGCR_ELO, AMCNTENCLR0_ELO, AMCNTENCLR1_ELO, AMCNTENSET0_ELO, AMCNTENSET1_ELO, AMCR_ELO, AMVCNTR0<n>_ELO, AMVCNTR1<n>_ELO, AMEVTYPE0<n>_ELO, and AMEVTYPE1<n>_ELO.

- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
  - AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVCNTR0<n>, AMEVCNTR1<n>, AMEVTYPE0<n>, and AMEVTYPE1<n>.

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 accesses to the activity monitors registers are trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0.</td>
</tr>
</tbody>
</table>

**Note**
- AMUSERENR_EL0 can always be read at EL0 and is not governed by this bit.

### Accessing the AMUSERENR_EL0

Accesses to this register use the following encodings:

**MRS <Xt>, AMUSERENR_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMUSERENR_EL0;
  endif
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AMUSERENR_EL0 = X[t];
    endif
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AMUSERENR_EL0 = X[t];
    endif
  elsif PSTATE.EL == EL3 then
    AMUSERENR_EL0 = X[t];
  endif
if PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return AMUSERENR_EL0;
  endif
else
  if PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AMUSERENR_EL0 = X[t];
    endif
  elsif PSTATE.EL == EL3 then
    AMUSERENR_EL0 = X[t];
  endif
if PSTATE.EL == EL0 then
  UNDEFINED;
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    AMUSERENR_EL0 = X[t];
  endif
else
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    AMUSERENR_EL0 = X[t];
  endif
```

**MSR AMUSERENR_EL0, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    AMUSERENR_EL0 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    AMUSERENR_EL0 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  AMUSERENR_EL0 = X[t];
```
The APDAKeyHi_EL1 characteristics are:

**Purpose**

Holds bits[127:64] of key A used for authentication of data pointer values.

**Note**

The term APDAKey_EL1 is used to describe the concatenation of APDAKeyHi_EL1: APDAKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APDAKeyHi_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APDAKeyHi_EL1 is a 64-bit register.

**Field descriptions**

The APDAKeyHi_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value |

**Bits [63:0]**

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APDAKeyHi_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, APDAKeyHi_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

Page 88
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.APDKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APDAKeyHi_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            APDAKeyHi_EL1 = X[t];
        elsif PSTATE.EL == EL3 then
            APDAKeyHi_EL1 = X[t];

MSR APDAKeyHi_EL1, <Xt>
APDAKeyLo_EL1, Pointer Authentication Key A for Data (bits[63:0])

The APDAKeyLo_EL1 characteristics are:

**Purpose**

Holds bits[63:0] of key A used for authentication of data pointer values.

**Note**

The term APDAKey_EL1 is used to describe the concatenation of APDAKeyHi_EL1: APDAKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APDAKeyLo_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APDAKeyLo_EL1 is a 64-bit register.

**Field descriptions**

The APDAKeyLo_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
</tr>
<tr>
<td>62</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
</tr>
<tr>
<td>31</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APDAKeyLo_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, APDAKeyLo_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.APDAKey == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return APDAKeyLo_EL1;
  end if
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    APDAKeyLo_EL1 = X[t];
  end if
elsif PSTATE.EL == EL3 then
  APDAKeyLo_EL1 = X[t];
else
  APDAKeyLo_EL1 = X[t];
end if

APDAKeyLo_EL1, Pointer Authentication Key A for Data (bits[63:0])

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

27/09/2019 18:22, 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The APDBKeyHi_EL1 characteristics are:

**Purpose**

Holds bits[127:64] of key B used for authentication of data pointer values.

**Note**

The term APDBKey_EL1 is used to describe the concatenation of APDBKeyHi_EL1: APDBKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APDBKeyHi_EL1 are undefined.

RW fields in this register reset to architecturally unknown values.

**Attributes**

APDBKeyHi_EL1 is a 64-bit register.

**Field descriptions**

The APDBKeyHi_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | 64 bit value, bits[127:64] of the 128 bit pointer authentication key value |

**Bits [63:0]**

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

This field resets to an architecturally unknown value.

**Accessing the APDBKeyHi_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, APDBKeyHi_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return APDBKeyHi_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APDBKeyHi_EL1;
    endif
elsif PSTATE.EL == EL3 then
    return APDBKeyHi_EL1;
endif

MSR APDBKeyHi_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        APDBKeyHi_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyHi_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    APDBKeyHi_EL1 = X[t];
endif

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
APDBKeyLo_EL1, Pointer Authentication Key B for Data (bits[63:0])

The APDBKeyLo_EL1 characteristics are:

**Purpose**

Holds bits[63:0] of key B used for authentication of data pointer values.

**Note**

The term APDBKey_EL1 is used to describe the concatenation of APDBKeyHi_EL1: APDBKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APDBKeyLo_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APDBKeyLo_EL1 is a 64-bit register.

**Field descriptions**

The APDBKeyLo_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value |

**Bits [63:0]**

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APDBKeyLo_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, APDBKeyLo_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && 
      HFGWTR_EL2.APDBKey == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return APDBKeyLo_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return APDBKeyLo_EL1;
  elsif PSTATE.EL == EL3 then
    return APDBKeyLo_EL1;

MSR APDBKeyLo_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && 
      HFGWTR_EL2.APDBKey == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    APDBKeyLo_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      APDBKeyLo_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    APDBKeyLo_EL1 = X[t];
APGAKeyHi_EL1, Pointer Authentication Key A for Code (bits[127:64])

The APGAKeyHi_EL1 characteristics are:

**Purpose**

Holds bits[127:64] of key used for generic pointer authentication code.

**Note**

The term APGAKey_EL1 is used to describe the concatenation of APGAKeyHi_EL1: APGAKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APGAKeyHi_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APGAKeyHi_EL1 is a 64-bit register.

**Field descriptions**

The APGAKeyHi_EL1 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value |

**Bits [63:0]**

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APGAKeyHi_EL1**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS &lt;Xt&gt;, APGAKeyHi_EL1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>op0</strong></td>
</tr>
<tr>
<td>0b11</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APGAKeyHi_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            APGAKeyHi_EL1 = X[t];
        elsif PSTATE.EL == EL3 then
            APGAKeyHi_EL1 = X[t];
else
    APGAKeyHi_EL1 = X[t];

MSR APGAKeyHi_EL1, <Xt>
APGAKeyLo_EL1, Pointer Authentication Key A for Code (bits[63:0])

The APGAKeyLo_EL1 characteristics are:

**Purpose**

Holds bits[63:0] of key used for generic pointer authentication code.

**Note**

The term APGAKey_EL1 is used to describe the concatenation of APGAKeyHi_EL1: APGAKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APGAKeyLo_EL1 are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

APGAKeyLo_EL1 is a 64-bit register.

**Field descriptions**

The APGAKeyLo_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
</tr>
<tr>
<td>31</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the APGAKeyLo_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, APGAKeyLo_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APGAKeyLo_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyLo_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    APGAKeyLo_EL1 = X[t];
else
    APGAKeyLo_EL1 = APGAKeyLo_EL1;
endif

MSR APGAKeyLo_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
APIKeyHi_EL1, Pointer Authentication Key A for Instruction (bits[127:64])

The APIKeyHi_EL1 characteristics are:

**Purpose**

Holds bits[127:64] of key A used for authentication of instruction pointer values.

**Note**

The term APIKey_EL1 is used to describe the concatenation of APIKeyHi_EL1: APIKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APIKeyHi_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APIKeyHi_EL1 is a 64-bit register.

**Field descriptions**

The APIKeyHi_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>64 bit value, bits[127:64] of the 128 bit pointer authentication key value</td>
</tr>
<tr>
<td>31</td>
<td>64 bit value, bits[127:64] of the 128 bit pointer authentication key value</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APIKeyHi_EL1**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.APIAKey == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return APIAKeyHi_EL1;
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return APIAKeyHi_EL1;
  end if;
elsif PSTATE.EL == EL3 then
  return APIAKeyHi_EL1;
end if;

APIAKeyHi_EL1, Pointer Authentication Key A for Instruction (bits[127:64])

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.APIAKey == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    APIAKeyHi_EL1 = X[t];
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    APIAKeyHi_EL1 = X[t];
  end if;
elsif PSTATE.EL == EL3 then
  APIAKeyHi_EL1 = X[t];
APIKeyLo_EL1, Pointer Authentication Key A for Instruction (bits[63:0])

The APIKeyLo_EL1 characteristics are:

**Purpose**

Holds bits[63:0] of key A used for authentication of instruction pointer values.

**Note**

The term APIKey_EL1 is used to describe the concatenation of APIKeyHi_EL1: APIKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APIKeyLo_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APIKeyLo_EL1 is a 64-bit register.

**Field descriptions**

The APIKeyLo_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APIKeyLo_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, APIKeyLo_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGWTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIAKeyLo_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyLo_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    APIAKeyLo_EL1 = X[t];
else
    APIAKeyLo_EL1 = X[t];

MSR APIAKeyLo_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
APIBKeyHi_EL1, Pointer Authentication Key B for Instruction (bits[127:64])

The APIBKeyHi_EL1 characteristics are:

**Purpose**

Holds bits[127:64] of key B used for authentication of instruction pointer values.

**Note**

The term APIBKey_EL1 is used to describe the concatenation of APIBKeyHi_EL1: APIBKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APIBKeyHi_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APIBKeyHi_EL1 is a 64-bit register.

**Field descriptions**

The APIBKeyHi_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

64 bit value, bits[127:64] of the 128 bit pointer authentication key value

Bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APIBKeyHi_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, APIBKeyHi_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIBKeyHi_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APIBKeyHi_EL1;
        elsif PSTATE.EL == EL3 then
            return APIBKeyHi_EL1;
else
    MSR APIBKeyHi_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyHi_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            APIBKeyHi_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        APIBKeyHi_EL1 = X[t];
APIBKeyLo_EL1, Pointer Authentication Key B for Instruction (bits[63:0])

The APIBKeyLo_EL1 characteristics are:

**Purpose**

Holds bits[63:0] of key B used for authentication of instruction pointer values.

**Note**

The term APIBKey_EL1 is used to describe the concatenation of APIBKeyHi_EL1: APIBKeyLo_EL1.

**Configuration**

This register is present only when ARMv8.3-PAuth is implemented. Otherwise, direct accesses to APIBKeyLo_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

APIBKeyLo_EL1 is a 64-bit register.

**Field descriptions**

The APIBKeyLo_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>64 bit value, bits[63:0] of the 128 bit pointer authentication key value</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

This field resets to an architecturally UNKNOWN value.

**Accessing the APIBKeyLo_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, APIBKeyLo_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIBKeyLo_EL1;
    elsif PSTATE_EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APIBKeyLo_EL1;
        elsif PSTATE_EL == EL3 then
            return APIBKeyLo_EL1;
    MSR APIBKeyLo_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyLo_EL1 = X[t];
    elsif PSTATE_EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.APK == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            APIBKeyLo_EL1 = X[t];
    elsif PSTATE_EL == EL3 then
        APIBKeyLo_EL1 = X[t];
AT S12E0R, Address Translate Stages 1 and 2 EL0 Read

The AT S12E0R characteristics are:

**Purpose**

Performs stage 1 and 2 address translations from EL0, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

**Configuration**

**Attributes**

AT S12E0R is a 64-bit System instruction.

**Field descriptions**

The AT S12E0R input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Input address for translation</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Input address for translation</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Input address for translation</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Input address for translation</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>Input address for translation</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S12E0R instruction**

Accesses to this instruction use the following encodings:

AT S12E0R, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
    AT_S1E0R(X[t]);
  else
    AT_S1E0R(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    AT_S1E0R(X[t]);
  elseif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
    AT_S1E0R(X[t]);
  else
    AT_S1E0R(X[t]);

AT S12E0W, Address Translate Stages 1 and 2 EL0 Write

The AT S12E0W characteristics are:

**Purpose**

Performs stage 1 and 2 address translations from EL0, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.[E2H, TGE] is not \{1, 1\}, the EL1&0 translation regime.
  - If HCR_EL2.[E2H, TGE] is \{1, 1\}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

**Configuration**

**Attributes**

AT S12E0W is a 64-bit System instruction.

**Field descriptions**

The AT S12E0W input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>Input address for translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60</td>
<td></td>
</tr>
<tr>
<td>59 58 57 56</td>
<td></td>
</tr>
<tr>
<td>55 54 53 52</td>
<td></td>
</tr>
<tr>
<td>51 50 49 48</td>
<td></td>
</tr>
<tr>
<td>47 46 45 44</td>
<td></td>
</tr>
<tr>
<td>43 42 41 40</td>
<td></td>
</tr>
<tr>
<td>39 38 37 36</td>
<td></td>
</tr>
<tr>
<td>35 34 33 32</td>
<td></td>
</tr>
<tr>
<td>31 30 29 28</td>
<td></td>
</tr>
<tr>
<td>27 26 25 24</td>
<td></td>
</tr>
<tr>
<td>23 22 21 20</td>
<td></td>
</tr>
<tr>
<td>19 18 17 16</td>
<td></td>
</tr>
<tr>
<td>15 14 13 12</td>
<td></td>
</tr>
<tr>
<td>11 10 9  8</td>
<td></td>
</tr>
<tr>
<td>7  6  5  4</td>
<td></td>
</tr>
<tr>
<td>3  2  1  0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S12E0W instruction**

Accesses to this instruction use the following encodings:

AT S12E0W, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
    AT_S1E0W(X[t]);
  else
    AT_S12E0W(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    AT_S1E0W(X[t]);
  elseif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
    AT_S1E0W(X[t]);
  else
    AT_S12E0W(X[t]);

AT S12E1R, Address Translate Stages 1 and 2 EL1 Read

The AT S12E1R characteristics are:

**Purpose**

Performs stage 1 and 2 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.(E2H, TGE) is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.(E2H, TGE) is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.(E2H, TGE) is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

**Configuration**

**Attributes**

AT S12E1R is a 64-bit System instruction.

**Field descriptions**

The AT S12E1R input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Input address for translation |
| Input address for translation |

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S12E1R instruction**

Accesses to this instruction use the following encodings:

AT S12E1R, <Xt>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x10);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AT_S1E1R(X[t]);
    else
        AT_S12E1R(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AT_S1E1R(X[t]);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AT_S1E1R(X[t]);
    else
        AT_S12E1R(X[t]);
AT S12E1W, Address Translate Stages 1 and 2 EL1 Write

The AT S12E1W characteristics are:

**Purpose**

Performs stage 1 and 2 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of HCR_EL2.{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

**Configuration**

**Attributes**

AT S12E1W is a 64-bit System instruction.

**Field descriptions**

The AT S12E1W input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S12E1W instruction**

Accesses to this instruction use the following encodings:

AT S12E1W, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
    AT_S1E1W(X[t]);
  else
    AT_S12E1W(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    AT_S1E1W(X[t]);
  elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
    AT_S1E1W(X[t]);
  else
    AT_S12E1W(X[t]);
AT S1E0R, Address Translate Stage 1 EL0 Read

The AT S1E0R characteristics are:

**Purpose**

Performs stage 1 address translation from EL0, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{ E2H, TGE } is not \( \{1, 1\} \), the EL1&0 translation regime.
  - If HCR_EL2.{ E2H, TGE } is \( \{1, 1\} \), the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

**Configuration**

**Attributes**

AT S1E0R is a 64-bit System instruction.

**Field descriptions**

The AT S1E0R input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is \( \text{RES} \).

**Executing the AT S1E0R instruction**

Accesses to this instruction use the following encodings:

\[
\text{AT S1E0R, } \langle Xt \rangle
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.ATS1E0R == '1' then
        AArch64.SystemAccessTrap(EL2, 0x10);
        else
            AT_S1E0R(X[t]);
    elsif PSTATE.EL == EL2 then
        AT_S1E0R(X[t]);
    elsif PSTATE.EL == EL3 then
        AT_S1E0R(X[t]);
AT S1E0W, Address Translate Stage 1 EL0 Write

The AT S1E0W characteristics are:

**Purpose**

Performs stage 1 address translation from EL0, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.(E2H, TGE) is not {1, 1}, the EL1&0 translation regime.
  - If HCR_EL2.(E2H, TGE) is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

**Configuration**

**Attributes**

AT S1E0W is a 64-bit System instruction.

**Field descriptions**

The AT S1E0W input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>Input address for translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>Input address for translation</td>
</tr>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S1E0W instruction**

Accesses to this instruction use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.E1 == EL0 then
  UNDEFINED;
elsif PSTATE.E1 == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x10);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTE == '1') &&
    HFGITR_EL2.ATS1E0W == '1' then
    AArch64.SystemAccessTrap(EL2, 0x10);
  else
    AT_S1E0W(X[t]);
  end
elsif PSTATE.E1 == EL2 then
  AT_S1E0W(X[t]);
elsif PSTATE.E1 == EL3 then
  AT_S1E0W(X[t]);
AT S1E1R, Address Translate Stage 1 EL1 Read

The AT S1E1R characteristics are:

**Purpose**

Performs stage 1 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of \( HCR\_EL2\).\{E2H, TGE\} is \{1, 1\}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of \( SCR\_EL3\).NS:
  - If \( HCR\_EL2\).\{E2H, TGE\} is not \{1, 1\}, the EL1&0 translation regime, accessed from EL1.
  - If \( HCR\_EL2\).\{E2H, TGE\} is \{1, 1\}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

**Configuration**

**Attributes**

AT S1E1R is a 64-bit System instruction.

**Field descriptions**

The AT S1E1R input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:0</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Input address for translation. The resulting address can be read from the \( PAR\_EL1\).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S1E1R instruction**

Accesses to this instruction use the following encodings:

AT S1E1R, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.ATS1E1R == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E1R(X[t]);
    elsif PSTATE.EL == EL2 then
        AT_S1E1R(X[t]);
    elsif PSTATE.EL == EL3 then
        AT_S1E1R(X[t]);

AT S1E1RP, Address Translate Stage 1 EL1 Read PAN

The AT S1E1RP characteristics are:

**Purpose**

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a read from a location will generate a permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.(E2H, TGE) is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.(E2H, TGE) is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

**Configuration**

This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to AT S1E1RP are UNDEFINED.

**Attributes**

AT S1E1RP is a 64-bit System instruction.

**Field descriptions**

The AT S1E1RP input value bit assignments are:

<table>
<thead>
<tr>
<th>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S1E1RP instruction**

Accesses to this instruction use the following encodings:

AT S1E1RP, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.ATS1E1RP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AT_S1E1RP(X[t]);
  end;
elsif PSTATE.EL == EL2 then
  AT_S1E1RP(X[t]);
elsif PSTATE.EL == EL3 then
  AT_S1E1RP(X[t]);
AT S1E1W, Address Translate Stage 1 EL1 Write

The AT S1E1W characteristics are:

**Purpose**

Performs stage 1 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of \( HCR_{EL2}.\{E2H, TGE\} \) is \( \{1, 1\} \), using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of \( SCR_{EL3}.NS \):
  - If \( HCR_{EL2}.\{E2H, TGE\} \) is not \( \{1, 1\} \), the EL1&0 translation regime, accessed from EL1.
  - If \( HCR_{EL2}.\{E2H, TGE\} \) is \( \{1, 1\} \), the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

**Configuration**

**Attributes**

AT S1E1W is a 64-bit System instruction.

**Field descriptions**

The AT S1E1W input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Input address for translation. The resulting address can be read from the \( PAR_{EL1} \).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is \( \text{RES}0 \).

**Executing the AT S1E1W instruction**

Accesses to this instruction use the following encodings:

\[
\text{AT S1E1W, } <\text{Xt}>
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.ATS1E1W == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AT_S1E1W(X[t]);
  end if;
elsif PSTATE.EL == EL2 then
  AT_S1E1W(X[t]);
elsif PSTATE.EL == EL3 then
  AT_S1E1W(X[t]);
AT S1E1WP, Address Translate Stage 1 EL1 Write PAN

The AT S1E1WP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a write to a location will generate a permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not \{1, 1\}, the EL1&0 translation regime, accessed from EL1.
  - If HCR_EL2.{E2H, TGE} is \{1, 1\}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to AT S1E1WP are UNDEFINED.

Attributes

AT S1E1WP is a 64-bit System instruction.

Field descriptions

The AT S1E1WP input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1WP instruction

Accesses to this instruction use the following encodings:

AT S1E1WP, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.ATS1E1WP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AT_S1E1WP(X[t]);
    end if;
elsif PSTATE.EL == EL2 then
    AT_S1E1WP(X[t]);
elsif PSTATE.EL == EL3 then
    AT_S1E1WP(X[t]);
AT S1E2R, Address Translate Stage 1 EL2 Read

The AT S1E2R characteristics are:

**Purpose**

Performs stage 1 address translation as defined for EL2, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

AT S1E2R is a 64-bit System instruction.

**Field descriptions**

The AT S1E2R input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S1E2R instruction**

Accesses to this instruction use the following encodings:

AT S1E2R, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
elseif PSTATE.EL == EL2 then
   AT_S1E2R(X[t]);
elsif PSTATE.EL == EL3 then
   if !EL2Enabled() then
      UNDEFINED;
   else
      AT_S1E2R(X[t]);

AT S1E2W, Address Translate Stage 1 EL2 Write

The AT S1E2W characteristics are:

**Purpose**

Performs stage 1 address translation as defined for EL2, with permissions as if writing to the given virtual address.

**Configuration**

**Attributes**

AT S1E2W is a 64-bit System instruction.

**Field descriptions**

The AT S1E2W input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S1E2W instruction**

Accesses to this instruction use the following encodings:

\[
\text{AT} \ S1E2W, \ <Xt>
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  AT_S1E2W(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    AT_S1E2W(X[t]);
```


The AT S1E3R characteristics are:

**Purpose**

Performs stage 1 address translation as defined for EL3, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

AT S1E3R is a 64-bit System instruction.

**Field descriptions**

The AT S1E3R input value bit assignments are:

```
|   63 |   62 |   61 |   60 |   59 |   58 |   57 |   56 |   55 |   54 |   53 |   52 |   51 |   50 |   49 |   48 |   47 |   46 |   45 |   44 |   43 |   42 |   41 |   40 |   39 |   38 |   37 |   36 |   35 |   34 |   33 |   32 |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| Input address for translation | Input address for translation |
|   31 |   30 |   29 |   28 |   27 |   26 |   25 |   24 |   23 |   22 |   21 |   20 |   19 |   18 |   17 |   16 |   15 |   14 |   13 |   12 |   11 |   10 |   9  |   8  |   7  |   6  |   5  |   4  |   3  |   2  |   1  |   0  |
```

**Bits [63:0]**

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

**Executing the AT S1E3R instruction**

Accesses to this instruction use the following encodings:

```
AT S1E3R, <Xt>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  AT_S1E3R(X[t]);
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
AT S1E3W, Address Translate Stage 1 EL3 Write

The AT S1E3W characteristics are:

Purpose

Performs stage 1 address translation as defined for EL3, with permissions as if writing to the given virtual address.

Configuration

Attributes

AT S1E3W is a 64-bit System instruction.

Field descriptions

The AT S1E3W input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Input address for translation

Input address for translation

Bits [63:0]

Input address for translation. The resulting address can be read from the PAR_EL1.

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E3W instruction

Accesses to this instruction use the following encodings:

AT S1E3W, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AT_S1E3W(X[t]);
CCSIDR2_EL1, Current Cache Size ID Register 2

The CCSIDR2_EL1 characteristics are:

**Purpose**

When ARMv8.3-CCIDX is implemented, provides the information about the architecture of the currently selected cache from bits[63:32] of CCSIDR_EL1.

When ARMv8.3-CCIDX is not implemented, this register is not implemented.

**Configuration**

AArch64 System register CCSIDR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register CCSIDR2[31:0].

This register is present only when ARMv8.3-CCIDX is implemented. Otherwise, direct accesses to CCSIDR2_EL1 are UNDEFINED.

If AArch32 is not implemented, it is IMPLEMENTATION DEFINED whether reading this register gives an UNKNOWN value or is UNDEFINED.

The implementation includes one CCSIDR2_EL1 for each cache that it can access. CSSELR_EL1 selects which Cache Size ID Register is accessible.

**Attributes**

CCSIDR2_EL1 is a 64-bit register.

**Field descriptions**

The CCSIDR2_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>NumSets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**NumSets, bits [23:0]**

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

**Accessing the CCSIDR2_EL1**

If CSSELR_EL1 Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR2_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR2_EL1 read is treated as NOP.
- The CCSIDR2_EL1 read is UNDEFINED.
- The CCSIDR2_EL1 read returns an UNKNOWN value.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return CCSIDR2_EL1;
  elsif PSTATE.EL == EL2 then
    return CCSIDR2_EL1;
  elsif PSTATE.EL == EL3 then
    return CCSIDR2_EL1;
CCSIDR_EL1, Current Cache Size ID Register

The CCSIDR_EL1 characteristics are:

**Purpose**

Provides information about the architecture of the currently selected cache.

**Configuration**

AArch64 System register CCSIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CCSIDR[31:0].

AArch64 System register CCSIDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register CCSIDR2.

The implementation includes one CCSIDR_EL1 for each cache that it can access. CSSELR_EL1 selects which Cache Size ID Register is accessible.

**Attributes**

CCSIDR_EL1 is a 64-bit register.

**Field descriptions**

The CCSIDR_EL1 bit assignments are:

**When ARMv8.3-CCIDX is implemented:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>NumSets</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td>Associativity</td>
<td>LineSize</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Note**

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

**Bits [63:56]**

Reserved, RES0.

**NumSets, bits [55:32]**

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

**Bits [31:24]**

Reserved, RES0.
Associativity, bits [23:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log₂(Number of bytes in cache line)) - 4. For example:

- For a line length of 16 bytes: Log₂(16) = 4, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: Log₂(32) = 5, LineSize entry = 1.

When ARMv8.5-MemTag is implemented and enabled, where a cache only holds Allocation tags, this field is RES0.

Otherwise:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RE0 | UNKOWN | NumSets | Associativity | LineSize |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:32]

Reserved, RE0.

Bits [31:28]

Reserved, UNKOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log₂(Number of bytes in cache line)) - 4. For example:

- For a line length of 16 bytes: Log₂(16) = 4, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: Log₂(32) = 5, LineSize entry = 1.

Accessing the CCSIDR_EL1

If CSSEL_EL1.Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR_EL1 read is treated as NOP.
• The CCSIDR_EL1 read is **UNDEFINED**.
• The CCSIDR_EL1 read returns an **UNKNOWN** value.

Accesses to this register use the following encodings:

**MRS <Xt>, CCSIDR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HFGRTR_EL2.CCSIDR_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return CCSIDR_EL1;
  elsif PSTATE.EL == EL2 then
    return CCSIDR_EL1;
  elsif PSTATE.EL == EL3 then
    return CCSIDR_EL1;
```
The CFP RCTX characteristics are:

### Purpose

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, control flow prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

**Note**

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

### Configuration

This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to CFP RCTX are UNDEFINED.

### Attributes

CFP RCTX is a 64-bit System instruction.

### Field descriptions

The CFP RCTX input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>GVMID</td>
<td>VMID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:49]**

Reserved, RES0.

**GVMID, bit [48]**

Execution of this instruction applies to all VMIDs or a specified VMID.
If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

**VMID, bits [47:32]**

Only applies when bit[48] is 0 and either:
- an EL1 context.
- an EL0 context when \( \text{HCR}\_\text{EL2}.E2H==0 \) or \( \text{HCR}\_\text{EL2}.TGE==0 \).

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and \( \text{HCR}\_\text{EL2}.E2H==0 \) or \( \text{HCR}\_\text{EL2}.TGE==0 \) then this field is treated as the current VMID.

When the instruction is executed at EL0 and \( \text{HCR}\_\text{EL2}.E2H==1 \) and \( \text{HCR}\_\text{EL2}.TGE==1 \) then this field is ignored.

**Bits [31:27]**

Reserved, RES0.

**NS, bit [26]**

Security State

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state</td>
</tr>
</tbody>
</table>

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

**EL, bits [25:24]**

Exception Level

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>EL0</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1</td>
</tr>
<tr>
<td>0b10</td>
<td>EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>EL3</td>
</tr>
</tbody>
</table>

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

**Bits [23:17]**

Reserved, RES0.

**GASID, bit [16]**

Execution of this instruction applies to all ASIDs or a specified ASID.

<table>
<thead>
<tr>
<th>GASID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>
If the instruction is executed at EL0, then this field has an Effective value of 0.

**ASID, bits [15:0]**

Only applies for an EL0 context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.

**Executing the CFP RCTX instruction**

Accesses to this instruction use the following encodings:

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b100</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.CFPRCTX == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  endif
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.CFPRCTX == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      else
        CFP_RCTX(X[t]);
      endif
    endif
  else
    if PSTATE.EL == EL2 then
      CFP_RCTX(X[t]);
    else
      if PSTATE.EL == EL3 then
        CFP_RCTX(X[t]);
      else
        CFP_RCTX(X[t]);
      endif
    endif
  endif
```
The CLIDR_EL1 characteristics are:

### Purpose

Identifies the type of cache, or caches, that are implemented at each level and can be managed using the architected cache maintenance instructions that operate by set/way, up to a maximum of seven levels. Also identifies the Level of Coherence (LoC) and Level of Unification (LoU) for the cache hierarchy.

### Configuration

AArch64 System register CLIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CLIDR[31:0].

### Attributes

CLIDR_EL1 is a 64-bit register.

### Field descriptions

The CLIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Ttype&lt;n&gt;, bits [2(n-1)+34:2(n-1)+33], for n = 1 to 7</td>
</tr>
<tr>
<td>62</td>
<td>Bits [63:47]</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>ICB, bits [32:30]</td>
</tr>
<tr>
<td>59</td>
<td>Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.</td>
</tr>
<tr>
<td>58</td>
<td>The possible values are:</td>
</tr>
<tr>
<td>57</td>
<td>0b00: No Tag Cache.</td>
</tr>
<tr>
<td>56</td>
<td>0b01: Separate Allocation Tag Cache.</td>
</tr>
<tr>
<td>55</td>
<td>0b10: Unified Allocation Tag and Data cache, Allocation Tags and Data in unified lines.</td>
</tr>
<tr>
<td>54</td>
<td>0b11: Unified Allocation Tag and Data cache, Allocation Tags and Data in separate lines.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Otherwise:</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>ICB, bits [32:30]</td>
</tr>
<tr>
<td>49</td>
<td>Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.</td>
</tr>
<tr>
<td>48</td>
<td>The possible values are:</td>
</tr>
</tbody>
</table>
ICB | Meaning
--- | ---
0b000 | Not disclosed by this mechanism.
0b001 | L1 cache is the highest Inner Cacheable level.
0b010 | L2 cache is the highest Inner Cacheable level.
0b011 | L3 cache is the highest Inner Cacheable level.
0b100 | L4 cache is the highest Inner Cacheable level.
0b101 | L5 cache is the highest Inner Cacheable level.
0b110 | L6 cache is the highest Inner Cacheable level.
0b111 | L7 cache is the highest Inner Cacheable level.

LoUU, bits [29:27]
Level of Unification Uniprocessor for the cache hierarchy.

LoC, bits [26:24]
Level of Coherence for the cache hierarchy.

LoUIS, bits [23:21]
Level of Unification Inner Shareable for the cache hierarchy.

Ctype<n>, bits [3(n-1)+2:3(n-1)], for n = 1 to 7
Cache Type fields. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy. Possible values of each field are:

<table>
<thead>
<tr>
<th>Ctype&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>No cache.</td>
</tr>
<tr>
<td>0b001</td>
<td>Instruction cache only.</td>
</tr>
<tr>
<td>0b010</td>
<td>Data cache only.</td>
</tr>
<tr>
<td>0b011</td>
<td>Separate instruction and data caches.</td>
</tr>
<tr>
<td>0b100</td>
<td>Unified cache.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If software reads the Cache Type fields from Ctype1 upwards, once it has seen a value of 000, no caches that can be managed using the architected cache maintenance instructions that operate by set/way exist at further-out levels of the hierarchy. So, for example, if Ctype3 is the first Cache Type field with a value of 000, the values of Ctype4 to Ctype7 must be ignored.

Accessing the CLIDR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, CLIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.CLIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CLIDR_EL1;
elsif PSTATE.EL == EL2 then
    return CLIDR_EL1;
else if PSTATE.EL == EL3 then
    return CLIDR_EL1;
CNTFRQ_EL0, Counter-timer Frequency register

The CNTFRQ_EL0 characteristics are:

**Purpose**

This register is provided so that software can discover the frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

**Configuration**

AArch64 System register CNTFRQ_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTFRQ[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTFRQ_EL0 is a 64-bit register.

**Field descriptions**

The CNTFRQ_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Clock frequency</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**Bits [31:0]**

Clock frequency. Indicates the system counter clock frequency, in Hz.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTFRQ_EL0**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS &lt;Xt&gt;, CNTFRQ_EL0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
</tr>
<tr>
<td>0b11</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
    CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      return CNTFRQ_EL0;
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
    CNTKCTL_EL2.<EL0PCTEN,EL0VCTEN> == '00' then
    return CNTFRQ_EL0;
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  end
  elsif PSTATE.EL == EL1 then
    return CNTFRQ_EL0;
  elseif PSTATE.EL == EL2 then
    return CNTFRQ_EL0;
  elseif PSTATE.EL == EL3 then
    return CNTFRQ_EL0;
endif

if IsHighestEL(PSTATE.EL) then
  CNTFRQ_EL0 = X[t];
else
  UNDEFINED;
end
CNTHCTL_EL2, Counter-timer Hypervisor Control register

The CNTHCTL_EL2 characteristics are:

**Purpose**

Controls the generation of an event stream from the physical counter, and access from EL1 to the physical counter and the EL1 physical timer.

**Configuration**

AArch64 System register CNTHCTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHCTL[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHCTL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHCTL_EL2 bit assignments are:

**When ARMv8.1-VHE is implemented and HCR_EL2.E2H == 1:**

| Bit Position | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Description  | RES0| EVNTIS| EL1NVVCT| EL1NPCT| EL1TVCT| EL1TVT| ECV| EL1PTEN| EL1PCTEN| EL0PTEN| EL0PTEN| EL0VCTEN| EL0PCTEN| EL0VCTEN| ELOVTE|
| Bits         | 31 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  |
| Values       | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  | 0  |

**Bits [63:18]**

Reserved, RES0.

**EVNTIS, bit [17]**

When ARMv8.6-ECV is implemented:

Controls the scale of the generation of the event stream.

<table>
<thead>
<tr>
<th>EVNTIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[15:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[23:8].</td>
</tr>
</tbody>
</table>

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**EL1NVVCT, bit [16]**

When ARMv8.6-ECV is implemented:

Traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1NVVCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If ( (HCR_EL2).E2H==1 &amp;&amp; (HCR_EL2).TGE==1)</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL\_EL2.ECV bit.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**EL1NVPCT, bit [15]**

When ARMv8.6-ECV is implemented:

Traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1NVPCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If ( (HCR_EL2).E2H==1 &amp;&amp; (HCR_EL2).TGE==1)</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL\_EL2.ECV bit.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**EL1TVCT, bit [14]**
When ARMv8.6-ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1TVCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>
| 0b1     | If HCR_EL2.\{E2H, TGE\} is \{1, 1\}, this control does not cause any instructions to be trapped.  
          If HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, then:  
          • In AArch64 state, traps EL0 and EL1 accesses to CNTVCT_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.EL0VTEN.  
          • In AArch32 state, traps EL0 and EL1 accesses to CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.EL0VTEN |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]

When ARMv8.6-ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1TVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>
| 0b1     | If HCR_EL2.\{E2H, TGE\} is \{1, 1\}, this control does not cause any instructions to be trapped.  
          If HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, then:  
          • In AArch64 state, traps EL0 and EL1 accesses to CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.EL0VTEN.  
          • In AArch32 state, traps EL0 and EL1 accesses to CNTV_CTL, CNTV_CVAL, and CNTV_TVAL to EL2, unless they are trapped by CNTKCTL_EL1.EL0VTEN or CNTKCTL.PL0VTEN |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
When ARMv8.6-ECV is implemented:

Enables the Enhanced Counter Virtualization functionality registers.

<table>
<thead>
<tr>
<th>ECV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Enhanced Counter Virtualization functionality is disabled.</td>
</tr>
<tr>
<td>01</td>
<td>When HCR_EL2.TGE == 1 or SCR_EL3.EEL2 == 1, then Enhanced Counter Virtualization functionality is disabled.</td>
</tr>
<tr>
<td></td>
<td>When SCR_EL3.NS or SCR_EL3.EEL2 are 1, and HCR_EL2.E2H or HCR_EL2.TGE are 0, then Enhanced Counter Virtualization functionality is enabled.</td>
</tr>
</tbody>
</table>
|      | This means that:
|      | • An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (PCount<63:0> - CNTPOFF_EL2<63:0>).               |
|      | • The EL1 physical timer interrupt is triggered when ((PCount<63:0> - CNTPOFF_EL2<63:0>) - PCVal<63:0>) is greater than or equal to 0. PCVal<63:0> is the EL1 physical timer compare value for this timer. |

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PTEN, bit [11]

When HCR_EL2.TGE is 0, traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>EL1PTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>From AArch64 state: EL0 and EL1 accesses to the CNTP_CTL_EL0, CNTP_CVAL_EL0, and CNTP_TVAL_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PTEN. From AArch32 state: EL0 and EL1 accesses to the CNTP_CTL, CNTP_CVAL, and CNTP_TVAL are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PTEN or CNTKCTL.PL0PTEN.</td>
</tr>
<tr>
<td>01</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [10]

When HCR_EL2.TGE is 0, traps EL0 and EL1 accesses to the EL1 physical counter register to EL2 when EL2 is enabled in the current Security state, as follows:

• In AArch64 state, accesses to CNTPCT_EL0 are trapped to EL2, reported using EC syndrome value 0x18.
• In AArch32 state, MRRC or MCRR accesses to CNTPCT are trapped to EL2, reported using EC syndrome value 0x04.
EL1PCTEN

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
</table>
| **0b0** From AArch64 state: EL0 and EL1 accesses to the CNTPCT_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1_EL0PCTEN.
From AArch32 state: EL0 and EL1 accesses to the CNTPCT are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1_EL0PCTEN or CNTKCTL_PLOPCTEN. |
| **0b1** This control does not cause any instructions to be trapped. |

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

ELOPTEN, bit [9]

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, traps EL0 accesses to the physical timer registers to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
</table>
| **0b0** EL0 using AArch64: EL0 accesses to the CNTP_CTL_EL0, CNTP_CVAL_EL0, and CNTP_TVAL_EL0 registers are trapped to EL2.
EL0 using AArch32: EL0 accesses to the CNTP_CTL, CNTP_CVAL, and CNTP_TVAL registers are trapped to EL2. |
| **0b1** This control does not cause any instructions to be trapped. |

This field resets to an architecturally UNKNOWN value.

ELOVTEN, bit [8]

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, traps EL0 accesses to the virtual timer registers to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
</table>
| **0b0** EL0 using AArch64: EL0 accesses to the CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0 registers are trapped to EL2.
EL0 using AArch32: EL0 accesses to the CNTV_CTL, CNTV_CVAL, and CNTV_TVAL registers are trapped to EL2. |
| **0b1** This control does not cause any instructions to be trapped. |

This field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit (0 to 15) of the counter register CNTPCT_EL0 is the trigger for the event stream generated from that counter, when that stream is enabled.

This field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the counter register CNTPCT_EL0 trigger bit, defined by EVNTI, generates an event when the event stream is enabled:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> A 0 to 1 transition of the trigger bit triggers an event.</td>
</tr>
<tr>
<td><strong>0b1</strong> A 1 to 0 transition of the trigger bit triggers an event.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
EVNTEN, bit [2]

Enables the generation of an event stream from the counter register CNTPCT_EL0:

<table>
<thead>
<tr>
<th>EVNTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables the event stream.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables the event stream.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

EL0VCTEN, bit [1]

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, traps EL0 accesses to the frequency register and virtual counter register to EL2.

<table>
<thead>
<tr>
<th>EL0VCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 using AArch64: EL0 accesses to the CNTVCT_EL0 are trapped to EL2.</td>
</tr>
<tr>
<td></td>
<td>EL0 using AArch64: EL0 accesses to the CNTFRQ_EL0 register are trapped to EL2, if CNTHCTL_EL2.EL0PCTEN is also 0.</td>
</tr>
<tr>
<td></td>
<td>EL0 using AArch32: EL0 accesses to the CNTVCT are trapped to EL2.</td>
</tr>
<tr>
<td></td>
<td>EL0 using AArch32: EL0 accesses to the CNTFRQ register are trapped to EL2, if CNTHCTL_EL2.EL0PCTEN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped.

When HCR_EL2.TGE is 1, traps EL0 accesses to the frequency register and physical counter register to EL2.

<table>
<thead>
<tr>
<th>EL0PCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 using AArch64: EL0 accesses to the CNTPCT_EL0 are trapped to EL2.</td>
</tr>
<tr>
<td></td>
<td>EL0 using AArch64: EL0 accesses to the CNTFRQ_EL0 register are trapped to EL2, if CNTHCTL_EL2.EL0PCTEN is also 0.</td>
</tr>
<tr>
<td></td>
<td>EL0 using AArch32: EL0 accesses to the CNTPCT are trapped to EL2.</td>
</tr>
<tr>
<td></td>
<td>EL0 using AArch32: EL0 accesses to the CNTFRQ and register are trapped to EL2, if CNTHCTL_EL2.EL0PCTEN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

The format applies in all Armv8.0 implementations, and it also contains a description of the behavior when EL3 is implemented and EL2 is not implemented.

Bits [63:18]

Reserved, RES0.
**EVNTIS, bit [17]**

*When ARMv8.6-ECV is implemented:* 

Controls the scale of the generation of the event stream.

<table>
<thead>
<tr>
<th>EVNTIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[15:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0[23:8].</td>
</tr>
</tbody>
</table>

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**EL1NVVCT, bit [16]**

*When ARMv8.6-ECV is implemented:* 

Traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1NVVCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If ((HCR_EL2.E2H==1 &amp; HCR_EL2.TGE==1)</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**EL1NVPCT, bit [15]**

*When ARMv8.6-ECV is implemented:* 

Traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.
If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**EL1TVCT, bit [14]**

When ARMv8.6-ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1TVCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If HCR_EL2.E2H == 1 &amp; HCR_EL2.TGE == 1</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**EL1TVT, bit [13]**

When ARMv8.6-ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state.

<table>
<thead>
<tr>
<th>EL1TVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, then: In AArch64 state, traps EL0 and EL1 accesses to CNTVCT_EL0 to EL2, unless they are trapped by CNTKCTL_EL1_EL0VCTEN. In AArch32 state, traps EL0 and EL1 accesses to CNTVCT to EL2, unless they are trapped by CNTKCTL_EL1_EL0VCTEN or CNTKCTL.PL0VCTEN.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
This control does not cause any instructions to be trapped.

If $HCR_{EL2}$.E2H is 0 or $HCR_{EL2}$.TGE is 0, then:
- In AArch64 state, traps EL0 and EL1 accesses to $CNTV\_CTL\_EL0$, $CNTV\_CVAL\_EL0$, and $CNTV\_TVAL\_EL0$ to EL2, unless they are trapped by $CNTKCTL\_EL1.ELOVTEN$.
- In AArch32 state, traps EL0 and EL1 accesses to $CNTV\_CTL$, $CNTV\_CVAL$, and $CNTV\_TVAL$ to EL2, unless they are trapped by $CNTKCTL\_EL1.ELOVTEN$ or $CNTKCTL\_PL0VTEN$.

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the $CNTHCTL\_EL2.ECV$ bit.

This field resets to an architecturally $UNKNOWN$ value.

$ECV$, bit [12]

When ARMv8.6-ECV is implemented:

Enables the Enhanced Counter Virtualization functionality registers.

This field resets to an architecturally $UNKNOWN$ value.

$EVNTI$, bits [7:4]

Selects which bit (0 to 15) of the counter register $CNTPCT\_EL0$ is the trigger for the event stream generated from that counter, when that stream is enabled.
This field resets to an architecturally **UNKNOWN** value.

**EVNTDIR, bit [3]**

Controls which transition of the counter register **CNTPCT_EL0** trigger bit, defined by EVNTI, generates an event when the event stream is enabled:

<table>
<thead>
<tr>
<th>EVNTDIR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A 0 to 1 transition of the trigger bit triggers an event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A 1 to 0 transition of the trigger bit triggers an event.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**EVNTEN, bit [2]**

Enables the generation of an event stream from the counter register **CNTPCT_EL0**:

<table>
<thead>
<tr>
<th>EVNTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables the event stream.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables the event stream.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**EL1PCEN, bit [1]**

Traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to **CNTP_CTL_EL0, CNTP_CVAL_EL0, CNTP_TVAL_EL0** are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 reported using EC syndrome value 0x3 and MRRC and MCRR accesses are trapped to EL2, reported using EC syndrome value 0x04:
  - **CNTP_CTL, CNTP_CVAL, CNTP_TVAL**.

<table>
<thead>
<tr>
<th>EL1PCEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>From AArch64 state: EL0 and EL1 accesses to the <strong>CNTP_CTL_EL0, CNTP_CVAL_EL0, CNTP_TVAL_EL0</strong> are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by <strong>CNTKCTL_EL1.EL0PTEN</strong>. From AArch32 state: EL0 and EL1 accesses to the <strong>CNTP_CTL, CNTP_CVAL, CNTP_TVAL</strong> are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by <strong>CNTKCTL_EL1.EL0PTEN</strong> or <strong>CNTKCTL.PL0PTEN</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

This field resets to an architecturally **UNKNOWN** value.

**EL1PCTEN, bit [0]**

Traps EL0 and EL1 accesses to the EL1 physical counter register to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to **CNTPCT_EL0** are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRRC or MCRR accesses to **CNTPCT** are trapped to EL2, reported using EC syndrome value 0x04.
<table>
<thead>
<tr>
<th>EL1PCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>From AArch64 state: EL0 and EL1 accesses to the CNTPCT_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PCTEN. From AArch32 state: EL0 and EL1 accesses to the CNTPCT are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

This field resets to an architecturally UNKNOWN value.

### Accessing the CNTHCTL_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHCTL_EL2 or CNTKCTL_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CNTHCTL_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return CNTHCTL_EL2;
elsif PSTATE.EL == EL3 then
  return CNTHCTL_EL2;
endif

**MSR CNTHCTL_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  CNTHCTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  CNTHCTL_EL2 = X[t];
endif

**MRS <Xt>, CNTKCTL_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTHCTL_EL2;
    else
        return CNTKCTL_EL1;
    elsif PSTATE.EL == EL3 then
        return CNTKCTL_EL1;

MSR CNTKCTL_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTHCTL_EL2 = X[t];
    else
        CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t];

CNGTHP_CTL_EL2, Counter-timer Hypervisor Physical Timer Control register

The CNGTHP_CTL_EL2 characteristics are:

**Purpose**

Control register for the EL2 physical timer.

**Configuration**

AArch64 System register CNGTHP_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNGTHP_CTL[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNGTHP_CTL_EL2 is a 64-bit register.

**Field descriptions**

The CNGTHP_CTL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>ISTATUS, bit [2]</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>ISTATUS, bit [2]</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>ISTATUS, bit [2]</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>56</td>
<td>ISTATUS, bit [2]</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [63:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see ‘Operation of the CompareValue views of the timers' and ‘Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:
IMASK

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from `CNTHP_TVAL_EL2` continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTHP_CTL_EL2**

When `HCR_EL2.E2H` is 1, without explicit synchronization, access from EL2 using the mnemonic `CNTHP_CTL_EL2` or `CNTP_CTL_EL0` are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CNTHP_CTL_EL2**

```assembly
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CTL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHP_CTL_EL2;
```

**MSR CNTHP_CTL_EL2, <Xt>**

```assembly
    op0  | op1  | CRn  | CRm  | op2  |
    ---- | ---- | ---- | ---- | ---- |
    0b11 | 0b100| 0b110| 0b0010| 0b001|
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elif PSTATE.EL == EL2 then
    CNTHP_CTL_EL2 = X[t];
elifs PSTATE.EL == EL3 then
    CNTHP_CTL_EL2 = X[t];
MRS <Xt>, CNTP_CTL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    endif
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    return CNTHPS_CTL_EL2;
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHP_CTL_EL2;
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHP_CTL_EL2;
else
    return CNTP_CTL_EL0;
e1f PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x180];
else
        return CNTP_CTL_EL0;
e1f PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHPS_CTL_EL2;
else HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
else
        return CNTP_CTL_EL0;
e1f PSTATE.EL == EL3 then
    return CNTP_CTL_EL0;
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
                AArch64.SystemAccessTrap(EL1, 0x18);
            else if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
                CNTHPS_CTL_EL2 = X[t];
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
                CNTHP_CTL_EL2 = X[t];
            else
                CNTP_CTL_EL0 = X[t];
            end if
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x180] = X[t];
        else
            CNTP_CTL_EL0 = X[t];
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
            CNTHPS_CTL_EL2 = X[t];
        elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            CNTHP_CTL_EL2 = X[t];
        else
            CNTP_CTL_EL0 = X[t];
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if HCR_EL2.E2H == '1' then
            CNTHPS_CTL_EL2 = X[t];
        elsif HCR_EL2.E2H == '0' then
            CNTHP_CTL_EL2 = X[t];
        else
            CNTP_CTL_EL0 = X[t];
        end if
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' then
                AArch64.SystemAccessTrap(EL1, 0x18);
            else
                if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
                    AArch64.SystemAccessTrap(EL2, 0x18);
                else
                    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
                        NVMem[0x180] = X[t];
                    else
                        CNTP_CTL_EL0 = X[t];
                    end if
                end if
            end if
        end if
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
            CNTHPS_CTL_EL2 = X[t];
        elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            CNTHP_CTL_EL2 = X[t];
        else
            CNTP_CTL_EL0 = X[t];
        end if
    elsif PSTATE.EL == EL3 then
        CNTP_CTL_EL0 = X[t];
    end if
CNTHP_CVAL_EL2, Counter-timer Physical Timer CompareValue register (EL2)

The CNTHP_CVAL_EL2 characteristics are:

**Purpose**

Holds the compare value for the EL2 physical timer.

**Configuration**

AArch64 System register CNTHP_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHP_CVAL[63:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHP_CVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHP_CVAL_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| CompareValue | CompareValue |

**CompareValue, bits [63:0]**

Holds the EL2 physical timer CompareValue.

When CNTHP_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHP_CTL_EL2.ISTATUS is set to 1.
- If CNTHP_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHP_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTHP_CVAL_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHP_CVAL_EL2 or CNTP_CVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, CNTHP_CVAL_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHP_CVAL_EL2;

MRS CNTHP_CVAL_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = X[t];

MRS <Xt>, CNTP_CVAL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x178];
        else
            return CNTP_CVAL_EL0;
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '0' then
                return CNTHPS_CVAL_EL2;
            elsif HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '1' then
                return CNTHP_CVAL_EL2;
            else
                return CNTP_CVAL_EL0;
            elsif PSTATE.EL == EL3 then
                return CNTP_CVAL_EL0;
            end
    end
end

MSR CNTP_CVAL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    endif
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHPS_CVAL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
    endif
else
    PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x178] = X[t];
        endif
    else
        CNTP_CVAL_EL0 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t];
The CNTHP_TVAL_EL2 characteristics are:

**Purpose**

Holds the timer value for the EL2 physical timer.

**Configuration**

AArch64 System register CNTHP_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHP_TVAL[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHP_TVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHP_TVAL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>TimerValue</td>
</tr>
<tr>
<td>31</td>
<td>TimerValue</td>
</tr>
<tr>
<td>30</td>
<td>TimerValue</td>
</tr>
<tr>
<td>29</td>
<td>TimerValue</td>
</tr>
<tr>
<td>28</td>
<td>TimerValue</td>
</tr>
<tr>
<td>27</td>
<td>TimerValue</td>
</tr>
<tr>
<td>26</td>
<td>TimerValue</td>
</tr>
<tr>
<td>25</td>
<td>TimerValue</td>
</tr>
<tr>
<td>24</td>
<td>TimerValue</td>
</tr>
<tr>
<td>23</td>
<td>TimerValue</td>
</tr>
<tr>
<td>22</td>
<td>TimerValue</td>
</tr>
<tr>
<td>21</td>
<td>TimerValue</td>
</tr>
<tr>
<td>20</td>
<td>TimerValue</td>
</tr>
<tr>
<td>19</td>
<td>TimerValue</td>
</tr>
<tr>
<td>18</td>
<td>TimerValue</td>
</tr>
<tr>
<td>17</td>
<td>TimerValue</td>
</tr>
<tr>
<td>16</td>
<td>TimerValue</td>
</tr>
<tr>
<td>15</td>
<td>TimerValue</td>
</tr>
<tr>
<td>14</td>
<td>TimerValue</td>
</tr>
<tr>
<td>13</td>
<td>TimerValue</td>
</tr>
<tr>
<td>12</td>
<td>TimerValue</td>
</tr>
<tr>
<td>11</td>
<td>TimerValue</td>
</tr>
<tr>
<td>10</td>
<td>TimerValue</td>
</tr>
<tr>
<td>9</td>
<td>TimerValue</td>
</tr>
<tr>
<td>8</td>
<td>TimerValue</td>
</tr>
<tr>
<td>7</td>
<td>TimerValue</td>
</tr>
<tr>
<td>6</td>
<td>TimerValue</td>
</tr>
<tr>
<td>5</td>
<td>TimerValue</td>
</tr>
<tr>
<td>4</td>
<td>TimerValue</td>
</tr>
<tr>
<td>3</td>
<td>TimerValue</td>
</tr>
<tr>
<td>2</td>
<td>TimerValue</td>
</tr>
<tr>
<td>1</td>
<td>TimerValue</td>
</tr>
<tr>
<td>0</td>
<td>TimerValue</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHP_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHP_CTL_EL2.ENABLE is 1, the value returned is (CNTHP_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHP_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHP_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHP_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHP_CTL_EL2.ISTATUS is set to 1.
- If CNTHP_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHP_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.
Accessing the CNTHP_TVAL_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHP_TVAL_EL2 or CNTP_TVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, CNTHP_TVAL_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td></td>
<td>0b110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return CNTHP_TVAL_EL2;
elsif PSTATE.EL == EL3 then
  return CNTHP_TVAL_EL2;

MSR CNTHP_TVAL_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td></td>
<td>0b110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  CNTHP_TVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  CNTHP_TVAL_EL2 = X[t];

MRS <Xt>, CNTP_TVAL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td></td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  endif
  elsif !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  endif
  elseif !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  endif
  elsif !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHPS_TVAL_EL2;
  elsif !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHP_TVAL_EL2;
  else
    return CNTP_TVAL_EL0;
  endif
elsif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return CNTP_TVAL_EL0;
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTHPS_TVAL_EL2;
  elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    return CNTHP_TVAL_EL2;
  else
    return CNTP_TVAL_EL0;
  endif
elsif PSTATE.EL == EL3 then
  return CNTP_TVAL_EL0;
endif

MSR CNTP_TVAL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b11</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHPS_TVAL_EL2 = X[t];
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_TVAL_EL2 = X[t];
      else
        CNTP_TVAL_EL0 = X[t];
      end
    end
  end
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' then
        CNTHPS_TVAL_EL2 = X[t];
      elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHP_TVAL_EL2 = X[t];
      else
        CNTP_TVAL_EL0 = X[t];
      end
    end
  else
    if PSTATE.EL == EL2 then
      if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_TVAL_EL2 = X[t];
      elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_TVAL_EL2 = X[t];
      else
        CNTP_TVAL_EL0 = X[t];
      end
    else
      if PSTATE.EL == EL3 then
        CNTP_TVAL_EL0 = X[t];
      end
    end
  end
The CNTHPS_CTL_EL2 characteristics are:

**Purpose**

Control register for the Secure EL2 physical timer.

**Configuration**

AArch64 System register CNTHPS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_CTL[31:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_CTL_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHPS_CTL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHPS_CTL_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|  RES0 |  ISTATUS | IMASK | ENABLE |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**Bits [63:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.
**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTHPS_TVAL_EL2** continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTHPS_CTL_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, CNTHPS_CTL_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) & SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() & HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) & SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    return CNTHPS_CTL_EL2;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    return CNTHPS_CTL_EL2;
else
  return CNTHPS_CTL_EL2;
```

**MSR CNTHPS_CTL_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>

Page 172
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t];

MRS <Xt>, CNTHPS_CTL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
                return CNTHPS_CTL_EL2;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
                return CNTHP_CTL_EL2;
            else
                return CNTP_CTL_EL0;
        endif
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    endif
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x180];
    else
        return CNTP_CTL_EL0;
    endif
else
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHPS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
    endif
else
    return CNTP_CTL_EL0;
endif

MSR CNTP_CTL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end if
  end if
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
  CNTHPS_CTL_EL2 = X[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
  CNTHP_CTL_EL2 = X[t];
else
  CNTP_CTL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
  NVMem[0x180] = X[t];
else
  CNTP_CTL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    CNTHPS_CTL_EL2 = X[t];
  elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    CNTHP_CTL_EL2 = X[t];
  end if
elsif PSTATE.EL == EL3 then
  CNTP_CTL_EL0 = X[t];
The CNTHPS_CVAL_EL2 characteristics are:

**Purpose**

Holds the compare value for the Secure EL2 physical timer.

**Configuration**

AArch64 System register CNTHPS_CVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_CVAL[31:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_CVAL_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHPS_CVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHPS_CVAL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>CompareValue</td>
</tr>
<tr>
<td>62</td>
<td>CompareValue</td>
</tr>
<tr>
<td>61</td>
<td>CompareValue</td>
</tr>
<tr>
<td>60</td>
<td>CompareValue</td>
</tr>
<tr>
<td>59</td>
<td>CompareValue</td>
</tr>
<tr>
<td>58</td>
<td>CompareValue</td>
</tr>
<tr>
<td>57</td>
<td>CompareValue</td>
</tr>
<tr>
<td>56</td>
<td>CompareValue</td>
</tr>
<tr>
<td>55</td>
<td>CompareValue</td>
</tr>
<tr>
<td>54</td>
<td>CompareValue</td>
</tr>
<tr>
<td>53</td>
<td>CompareValue</td>
</tr>
<tr>
<td>52</td>
<td>CompareValue</td>
</tr>
<tr>
<td>51</td>
<td>CompareValue</td>
</tr>
<tr>
<td>50</td>
<td>CompareValue</td>
</tr>
<tr>
<td>49</td>
<td>CompareValue</td>
</tr>
<tr>
<td>48</td>
<td>CompareValue</td>
</tr>
<tr>
<td>47</td>
<td>CompareValue</td>
</tr>
<tr>
<td>46</td>
<td>CompareValue</td>
</tr>
<tr>
<td>45</td>
<td>CompareValue</td>
</tr>
<tr>
<td>44</td>
<td>CompareValue</td>
</tr>
<tr>
<td>43</td>
<td>CompareValue</td>
</tr>
<tr>
<td>42</td>
<td>CompareValue</td>
</tr>
<tr>
<td>41</td>
<td>CompareValue</td>
</tr>
<tr>
<td>40</td>
<td>CompareValue</td>
</tr>
<tr>
<td>39</td>
<td>CompareValue</td>
</tr>
<tr>
<td>38</td>
<td>CompareValue</td>
</tr>
<tr>
<td>37</td>
<td>CompareValue</td>
</tr>
<tr>
<td>36</td>
<td>CompareValue</td>
</tr>
<tr>
<td>35</td>
<td>CompareValue</td>
</tr>
<tr>
<td>34</td>
<td>CompareValue</td>
</tr>
<tr>
<td>33</td>
<td>CompareValue</td>
</tr>
<tr>
<td>32</td>
<td>CompareValue</td>
</tr>
<tr>
<td>31</td>
<td>CompareValue</td>
</tr>
<tr>
<td>30</td>
<td>CompareValue</td>
</tr>
<tr>
<td>29</td>
<td>CompareValue</td>
</tr>
<tr>
<td>28</td>
<td>CompareValue</td>
</tr>
<tr>
<td>27</td>
<td>CompareValue</td>
</tr>
<tr>
<td>26</td>
<td>CompareValue</td>
</tr>
<tr>
<td>25</td>
<td>CompareValue</td>
</tr>
<tr>
<td>24</td>
<td>CompareValue</td>
</tr>
<tr>
<td>23</td>
<td>CompareValue</td>
</tr>
<tr>
<td>22</td>
<td>CompareValue</td>
</tr>
<tr>
<td>21</td>
<td>CompareValue</td>
</tr>
<tr>
<td>20</td>
<td>CompareValue</td>
</tr>
<tr>
<td>19</td>
<td>CompareValue</td>
</tr>
<tr>
<td>18</td>
<td>CompareValue</td>
</tr>
<tr>
<td>17</td>
<td>CompareValue</td>
</tr>
<tr>
<td>16</td>
<td>CompareValue</td>
</tr>
<tr>
<td>15</td>
<td>CompareValue</td>
</tr>
<tr>
<td>14</td>
<td>CompareValue</td>
</tr>
<tr>
<td>13</td>
<td>CompareValue</td>
</tr>
<tr>
<td>12</td>
<td>CompareValue</td>
</tr>
<tr>
<td>11</td>
<td>CompareValue</td>
</tr>
<tr>
<td>10</td>
<td>CompareValue</td>
</tr>
<tr>
<td>9</td>
<td>CompareValue</td>
</tr>
<tr>
<td>8</td>
<td>CompareValue</td>
</tr>
<tr>
<td>7</td>
<td>CompareValue</td>
</tr>
<tr>
<td>6</td>
<td>CompareValue</td>
</tr>
<tr>
<td>5</td>
<td>CompareValue</td>
</tr>
<tr>
<td>4</td>
<td>CompareValue</td>
</tr>
<tr>
<td>3</td>
<td>CompareValue</td>
</tr>
<tr>
<td>2</td>
<td>CompareValue</td>
</tr>
<tr>
<td>1</td>
<td>CompareValue</td>
</tr>
<tr>
<td>0</td>
<td>CompareValue</td>
</tr>
</tbody>
</table>

**CompareValue, bits [63:0]**

Holds the EL2 physical timer CompareValue.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTHPS_CVAL_EL2**

Accesses to this register use the following encodings:
MRS <Xt>, CNTHPS_CVAL_EL2

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b100 & 0b1110 & 0b0101 & 0b010 \\
\hline
\end{array}
\]

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
else
    return CNTHPS_CVAL_EL2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t];
    return CNTHPS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t];
    return CNTHPS_CVAL_EL2;
MSR CNTHPS_CVAL_EL2, <Xt>

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b100 & 0b1110 & 0b0101 & 0b010 \\
\hline
\end{array}
\]

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
else
    return CNTHPS_CVAL_EL2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t];
else
    return CNTHPS_CVAL_EL2;
MSR <Xt>, CNTP_CVAL_EL0

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b011 & 0b1110 & 0b0010 & 0b010 \\
\hline
\end{array}
\]
if PSTATE_EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
      return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
      return CNTHP_CVAL_EL2;
    else
      return CNTP_CVAL_EL0;
  else
    if PSTATE_EL == EL1 then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x178];
      else
        return CNTP_CVAL_EL0;
    elsif PSTATE_EL == EL2 then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHCTL_EL2.EL0PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x178];
      else
        return CNTP_CVAL_EL0;
    elsif PSTATE_EL == EL3 then
      return CNTP_CVAL_EL0;
  else
    return CNTP_CVAL_EL0;

MSR CNTP_CVAL_EL0, <Xt>
if PSTATE_EL == EL0 then
    if !ELUsingAArch32(EL1) && !ELEnabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x178] = X[t];
        endif
    endif
    CNTP_CVAL_EL0 = X[t];
else
    CNTP_CVAL_EL0 = X[t];
endif
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        NVMem[0x178] = X[t];
    endif
    CNTP_CVAL_EL0 = X[t];
else
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_CVAL_EL2 = X[t];
    endif
    CNTP_CVAL_EL0 = X[t];
else
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    endif
    CNTP_CVAL_EL0 = X[t];
endif
The CNTHPS_TVAL_EL2 characteristics are:

**Purpose**

Holds the timer value for the Secure EL2 physical timer.

**Configuration**

AArch64 System register CNTHPS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHPS_TVAL[31:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_TVAL_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHPS_TVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHPS_TVAL_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|    |
| RES0 | TimerValue |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHPS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHPS_CTL_EL2.ENABLE is 1, the value returned is (CNTHPS_CVAL_EL2 - CNTPCT_EL0).

On a write of this register, CNTHPS_CVAL_EL2 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTHPS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.
Accessing the CNTHPS_TVAL_EL2

Accesses to this register use the following encodings:

**MRS <Xt>, CNTHPS_TVAL_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    return CNTHPS_TVAL_EL2;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    CNTHPS_TVAL_EL2 = X[t];
endif

**MSR CNTHPS_TVAL_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    CNTHPS_TVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    CNTHPS_TVAL_EL2 = X[t];
endif

**MRS <Xt>, CNTVAL_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  endif
endif

if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return CNTHPS_TVAL_EL2;
    endif
  endif
endif

if PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTHPS_TVAL_EL2;
  else
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
      return CNTHP_TVAL_EL2;
    else
      return CNTP_TVAL_EL0;
    endif
  endif
endif

if PSTATE.EL == EL3 then
  return CNTP_TVAL_EL0;
endif

MSR CNTP_TVAL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !EL2Enabled() && HCR_EL2.E2H.TGE == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end if
    end if
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL0PTEN == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    else
        AArch64.SystemAccessTrap(EL2, 0x18);
    end if
end if

if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H.TGE == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H.TGE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    end if
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL1PTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    end if
end if

if PSTATE.EL == EL2 then
    if HCR_EL2.E2H.TGE == '1' && SCR_EL3.NS == '0' then
        CNTHPS_TVAL_EL2 = X[t];
    elseif HCR_EL2.E2H.TGE == '1' && SCR_EL3.NS == '1' then
        CNTHP_TVAL_EL2 = X[t];
    else
        CNTP_TVAL_EL0 = X[t];
    end if
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EL1PCEN == '1' then
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end if
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H.TGE == '0' && SCR_EL3.NS == '0' then
            CNTHPS_TVAL_EL2 = X[t];
        elseif HCR_EL2.E2H.TGE == '0' && SCR_EL3.NS == '1' then
            CNTHP_TVAL_EL2 = X[t];
        else
            CNTP_TVAL_EL0 = X[t];
        end if
    else
        if PSTATE.EL == EL3 then
            CNTP_TVAL_EL0 = X[t];
        end if
    end if

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The CNTHV_CTL_EL2 characteristics are:

**Purpose**

Control register for the EL2 virtual timer.

**Configuration**

AArch64 System register CNTHV_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHV_CTL[31:0].

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CNTHV_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHV_CTL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHV_CTL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.
**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTHV_TVAL_EL2** continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTHV_CTL_EL2**

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL2 using the mnemonic **CNTHV_CTL_EL2** or **CNTV_CTL_EL0** are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CNTHV_CTL_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHV_CTL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHV_CTL_EL2;

**MSR CNTHV_CTL_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsesif PSTATE.EL == EL2 then
  CNTV_CTL_EL2 = X[t];
elsesif PSTATE.EL == EL3 then
  CNTV_CTL_EL2 = X[t];

MRS <Xt>, CNTV_CTL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  else
    return CNTV_CTL_EL0;
  endif
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    return CNTV_CTL_EL2;
  else
    return CNTV_CTL_EL0;
  endif
elseif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTV_CTL_EL2;
  else
    return CNTV_CTL_EL0;
  endif
elseif PSTATE.EL == EL3 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    return CNTV_CTL_EL2;
  else
    return CNTV_CTL_EL0;
  endif
else
  return CNTV_CTL_EL0;
endif

MSR CNTV_CTL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    endif
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        CNTHVS_CTL_EL2 = X[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHV_CTL_EL2 = X[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x170] = X[t];
        else
            CNTV_CTL_EL0 = X[t];
        endif
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
            CNTHS_CTL_EL2 = X[t];
        elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            CNTHV_CTL_EL2 = X[t];
        else
            CNTV_CTL_EL0 = X[t];
        endif
    elseif PSTATE.EL == EL3 then
        CNTV_CTL_EL0 = X[t];
    endif
end if
The CNTHV_CVAL_EL2 characteristics are:

**Purpose**

Holds the compare value for the EL2 virtual timer.

**Configuration**

AArch64 System register CNTHV_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTHV_CVAL[63:0].

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CNTHV_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHV_CVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHV_CVAL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**CompareValue, bits [63:0]**

Holds the EL2 virtual timer CompareValue.

When CNTHV_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHV_CTL_EL2.ISTATUS is set to 1.
- If CNTHV_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHV_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTHV_CVAL_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHV_CVAL_EL2 or CNTV_CVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other mnemonic.
Accesses to this register use the following encodings:

**MRS <Xt>, CNTHV_CVAL_EL2**

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHV_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHV_CVAL_EL2;

**MSR CNTHV_CVAL_EL2, <Xt>**

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = <Xt>
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = <Xt>

**MRS <Xt>, CNTV_CVAL_EL0**

```
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHVS_CVAL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHV_CVAL_EL2;
  else
    return CNTV_CVAL_EL0;
  elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
      return NVMem[0x168];
    else
      return CNTV_CVAL_EL0;
  elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
      return CNTHVS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
      return CNTHV_CVAL_EL2;
    else
      return CNTV_CVAL_EL0;
  elseif PSTATE.EL == EL3 then
    return CNTV_CVAL_EL0;
else
  MSR CNTV_CVAL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  end
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
  CNTHVS_CVAL_EL2 = X[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
  CNTHV_CVAL_EL2 = X[t];
else
  CNTV_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x168] = X[t];
  else
    CNTV_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    CNTHVS_CVAL_EL2 = X[t];
elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    CNTHV_CVAL_EL2 = X[t];
else
    CNTV_CVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
  CNTV_CVAL_EL0 = X[t];
CNTHV_TVAL_EL2, Counter-timer Virtual Timer TimerValue Register (EL2)

The CNTHV_TVAL_EL2 characteristics are:

**Purpose**

Holds the timer value for the EL2 virtual timer.

**Configuration**

AArch64 System register CNTHV_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHV_TVAL[31:0].

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CNTHV_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHV_TVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHV_TVAL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>TimerValue</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>RES0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHV_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHV_CTL_EL2.ENABLE is 1, the value returned is (CNTHV_CVAL_EL2 - CNTVCT_EL0).

On a write of this register, CNTHV_CVAL_EL2 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHV_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CNTHV_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHV_CTL_EL2.ISTATUS is set to 1.
- If CNTHV_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHV_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count, so the TimerValue view appears to continue to count down.
This field resets to an architecturally **UNKNOWN** value.

## Accessing the CNTHV_TVAL_EL2

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL2 using the mnemonic `CNTHV_TVAL_EL2` or `CNTV_TVAL_EL0` are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

### MRS <Xt>, CNTHV_TVAL_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHV_TVAL_EL2;
else
    return CNTHV_TVAL_EL2;
```

### MSR CNTHV_TVAL_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_TVAL_EL2 = X[t];
else
    CNTHV_TVAL_EL2 = X[t];
```

### MRS <Xt>, CNTV_TVAL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && ELUsingAArch32(EL2) then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL_EL0;
    end if
else
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL1 then
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL_EL0;
    end if
else
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_TVAL_EL0;
end if

MSR CNTV_TVAL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && ELUsingAArch32(EL2) then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL_EL0;
    end if
else
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL1 then
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL_EL0;
    end if
else
    return CNTV_TVAL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_TVAL_EL0;
end if
CNTHVS_CTL_EL2, Counter-timer Secure Virtual Timer Control register (EL2)

The CNTHVS_CTL_EL2 characteristics are:

**Purpose**

Control register for the Secure EL2 virtual timer.

**Configuration**

AArch64 System register CNTHVS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHVS_CTL[31:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_CTL_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHVS_CTL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHVS_CTL_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the CNTHVS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.
IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the CNTHVS_CTL_EL2.ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHVS_TVAL_EL2 continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTHVS_CTL_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, CNTHVS_CTL_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
    else
        return CNTHVS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return CNTHVS_CTL_EL2;
else
    MSR CNTHVS_CTL_EL2, <Xt>
```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    CNTHVS_CTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    CNTHVS_CTL_EL2 = X[t];

MRS <Xt>, CNTV_CTL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b11</td>
<td>0b001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    return CNTHVS_CTL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHV_CTL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHV_CTL_EL2;
  else
    return CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '11' then
    return NVMem[0x170];
  else
    return CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTHVS_CTL_EL2;
  elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    return CNTHV_CTL_EL2;
  else
    return CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
  return CNTHV_CTL_EL2;
else
  return CNTV_CTL_EL0;

MSR CNTV_CTL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b11</td>
<td>0b001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) & & !(EL2Enabled() & & HCR_EL2.<E2H,TGE> == '11') & & CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  elseif EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.<E2H,TGE> == '11' & & CNTKCTL_EL2.EL0VTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.<E2H,TGE> == '11' & & SCR_EL3.NS == '0' then
    CNTHVS_CTL_EL2 = X[t];
  elseif EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.<E2H,TGE> == '11' & & SCR_EL3.NS == '1' then
    CNTHV_CTL_EL2 = X[t];
  else
    CNTV_CTL_EL0 = X[t];
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x170] = X[t];
  else
    CNTV_CTL_EL0 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' & & SCR_EL3.NS == '0' then
    CNTHVS_CTL_EL2 = X[t];
  elseif HCR_EL2.E2H == '1' & & SCR_EL3.NS == '1' then
    CNTHV_CTL_EL2 = X[t];
  else
    CNTV_CTL_EL0 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  CNTV_CTL_EL0 = X[t];
endif
The CNTHS_CVALID_EL2 characteristics are:

### Purpose

Holds the compare value for the Secure EL2 virtual timer.

### Configuration

AArch64 System register CNTHS_CVALID_EL2 bits [63:0] are architecturally mapped to AArch32 System register `CNTHS_CVALID[63:0]`.

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to `CNTHS_CVALID_EL2` are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

CNTHS_CVALID_EL2 is a 64-bit register.

### Field descriptions

The CNTHS_CVALID_EL2 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**CompareValue**, bits [63:0]

Holds the Secure EL2 virtual timer CompareValue.

When `CNTHS_CTL_EL2`.ENABLE is 1, the timer condition is met when `(CNTVCT_EL0 - CompareValue)` is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- `CNTHS_CTL_EL2`.ISTATUS is set to 1.
- If `CNTHS_CTL_EL2`.IMASK is 0, an interrupt is generated.

When `CNTHS_CTL_EL2`.ENABLE is 0, the timer condition is not met, but `CNTVCT_EL0` continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

### Accessing the CNTHS_CVALID_EL2

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else
  return CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    CNTHVS_CVAL_EL2 = X[t];
  return CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    CNTHVS_CVAL_EL2 = X[t];
else
  return CNTHVS_CVAL_EL2;

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else
  return CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    CNTHVS_CVAL_EL2 = X[t];
else
  return CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
else
  CNTHVS_CVAL_EL2 = X[t];
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x168];
        else
            return CNTV_CVAL_EL0;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
            return CNTHVS_CVAL_EL2;
        elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            return CNTHV_CVAL_EL2;
        else
            return CNTV_CVAL_EL0;
    elsif PSTATE.EL == EL3 then
        return CNTV_CVAL_EL0;
end if

MSR CNTV_CVAL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTV_CVAL_EL2 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
            NVMem[0x168] = X[t];
        else
            CNTV_CVAL_EL0 = X[t];
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
                CNTHVS_CVAL_EL2 = X[t];
            elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
                CNTHV_CVAL_EL2 = X[t];
            else
                CNTV_CVAL_EL0 = X[t];
            end
        elsif PSTATE.EL == EL3 then
            CNTV_CVAL_EL0 = X[t];
        end
    end
end

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The CNTHVS_TVAL_EL2 characteristics are:

**Purpose**

Holds the timer value for the Secure EL2 virtual timer.

**Configuration**

AArch64 System register CNTHVS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register CNTHVS_TVAL[31:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_TVAL_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHVS_TVAL_EL2 is a 64-bit register.

**Field descriptions**

The CNTHVS_TVAL_EL2 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | TimerValue |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHVS_CTL_EL2.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHVS_CTL_EL2.ENABLE is 1, the value returned is (CNTHVS_CVAL_EL2 - CNTVCT_EL0).

On a write of this register, CNTHVS_CVAL_EL2 is set to (CNTVCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHVS_CTL_EL2.ENABLE is 1, the timer condition is met when ((CNTVCT_EL0 - CNTHVS_CVAL_EL2) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHVS_CTL_EL2.ISTATUS is set to 1.
- If CNTHVS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.
Accessing the CNTHVS_TVAL_EL2

Accesses to this register use the following encodings:

MRS <Xt>, CNTHVS_TVAL_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elses PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
elses EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
elses
        UNDEFINED;
elses PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
elses
        return CNTHVS_TVAL_EL2;
elses PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
elses
        return CNTHVS_TVAL_EL2;

MSR CNTHVS_TVAL_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elses PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
elses EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
elses
        UNDEFINED;
elses PSTATE.EL == EL2 then
    if HaveEL(EL3) && SCR_EL3.NS == '1' then
        UNDEFINED;
elses
        CNTHVS_TVAL_EL2 = X[t];
elses PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
elses
        CNTHVS_TVAL_EL2 = X[t];

MRS <Xt>, CNTV_TVAL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL1.EL0VTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    CNTV_TVAL_EL2 = X[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    CNTV_TVAL_EL2 = X[t];
  else
    CNTV_TVAL_EL0 = X[t];
  else
    CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
  CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    CNTV_TVAL_EL2 = X[t];
  elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    CNTV_TVAL_EL2 = X[t];
  else
    CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
  CNTV_TVAL_EL0 = X[t];
else
  CNTV_TVAL_EL0 = X[t];

MSR CNTV_TVAL_EL0, <Xt>
CNTKCTL_EL1, Counter-timer Kernel Control register

The CNTKCTL_EL1 characteristics are:

**Purpose**

When ARMv8.1-VHE is not implemented, or when HCR_EL2.{E2H, TGE} is not {1, 1}, this register controls the generation of an event stream from the virtual counter, and access from EL0 to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this register does not cause any event stream from the virtual counter to be generated, and does not control access to the counters and timers. The access to counters and timers at EL0 is controlled by CNTHCTL_EL2.

**Configuration**

AArch64 System register CNTKCTL_EL1 bits [31:0] are architecturally mapped to AArch32 System register CNTKCTL[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTKCTL_EL1 is a 64-bit register.

**Field descriptions**

The CNTKCTL_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>RES0</th>
<th>EVNTIS</th>
<th>RES0</th>
<th>ELOPTEN</th>
<th>ELOVTEN</th>
<th>EVNTI</th>
<th>EVNTDIR</th>
<th>EVNTEN</th>
<th>ELOVCTEN</th>
<th>ELOPCTEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-18</td>
<td>17</td>
<td>16151413121110</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>16-0</td>
<td>3130292827262524232221201918</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:18]**

Reserved, RES0.

**EVNTIS, bit [17]**

When ARMv8.6-ECV is implemented:

Controls the scale of the generation of the event stream.

<table>
<thead>
<tr>
<th>EVNTIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CNTKCTL_EL1.EVNTI field applies to CNTVCT_EL0[15:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>The CNTKCTL_EL1.EVNTI field applies to CNTVCT_EL0[23:8].</td>
</tr>
</tbody>
</table>

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
Bits [16:10]

Reserved, RES0.

**ELOPTEN, bit [9]**

Traps EL0 accesses to the physical timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, the following registers are trapped, reported using EC syndrome value 0x18:
  - CNTP_CTL_EL0, CNTP_CVAL_EL0, and CNTP_TVAL_EL0.

- In AArch32 state, MRC and MCR accesses to the following registers are trapped, reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped, reported using EC syndrome value 0x04:
  - CNTP_CTL, CNTP_CVAL, CNTP_TVAL.

<table>
<thead>
<tr>
<th>ELOPTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 accesses to the physical timer registers are trapped to EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

This field resets to an architecturally **UNKNOWN** value.

**EL0VTEN, bit [8]**

Traps EL0 accesses to the virtual timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped, reported using EC syndrome value 0x18:
  - CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0.

- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped using EC syndrome value 0x04:
  - CNTV_CTL, CNTV_CVAL, and CNTV_TVAL.

<table>
<thead>
<tr>
<th>EL0VTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 accesses to the virtual timer registers are trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented and HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

This field resets to an architecturally **UNKNOWN** value.

**EVNTI, bits [7:4]**

Selects which bit (0 to 15) of the counter register CNTVCT_EL0 is the trigger for the event stream generated from that counter, when that stream is enabled.

This field resets to an architecturally **UNKNOWN** value.

**EVNTDIR, bit [3]**

Controls which transition of the counter register CNTVCT_EL0 trigger bit, defined by EVNTI, generates an event when the event stream is enabled:

<table>
<thead>
<tr>
<th>EVNTDIR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A 0 to 1 transition of the trigger bit triggers an event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A 1 to 0 transition of the trigger bit triggers an event.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally **UNKNOWN** value.

### EVNTEN, bit [2]

When ARMv8.1-VHE is not implemented, or when \texttt{HCR\_EL2}.\{E2H, TGE\} is not \{1, 1\}, enables the generation of an event stream from the counter register \texttt{CNTVCT\_EL0}:

<table>
<thead>
<tr>
<th>EVNTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables the event stream.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables the event stream.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented and \texttt{HCR\_EL2}.\{E2H, TGE\} is \{1, 1\}, this control does not enable the event stream.

This field resets to 0.

### ELOVCTEN, bit [1]

Traps EL0 accesses to the frequency register and virtual counter register to EL1, or to EL2 when it is implemented and enabled for the current Security state and \texttt{HCR\_EL2}.TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped and reported using EC syndrome value 0x18:
  - \texttt{CNTVCT\_EL0} and if \texttt{CNTKCTL\_EL1}.EL0VCTEN is 0, \texttt{CNTFRQ\_EL0}.
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
  - \texttt{CNTVCT} and if \texttt{CNTKCTL\_EL1}.EL0VCTEN is 0, \texttt{CNTFRQ}.

<table>
<thead>
<tr>
<th>ELOVCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 accesses to the frequency register and virtual counter registers are trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented and \texttt{HCR\_EL2}.\{E2H, TGE\} is \{1, 1\}, this control does not cause any instructions to be trapped.

This field resets to an architecturally **UNKNOWN** value.

### ELOPCTEN, bit [0]

Traps EL0 accesses to the frequency register and physical counter register to EL1, or to EL2 when it is implemented and enabled for the current Security state and \texttt{HCR\_EL2}.TGE is 1, as follows:

- In AArch64 state, the following registers are trapped, reported using EC syndrome value 0x18:
  - \texttt{CNTPCT\_EL0} and if \texttt{CNTKCTL\_EL1}.EL0PCTEN is 0, \texttt{CNTFRQ\_EL0}.
- In AArch32 state, MCR or MRC accesses the following registers are trapped, reported using EC syndrome value 0x03, MCRR or MRRC accesses are trapped and reported using EC syndrome value 0x04:
  - \texttt{CNTCT} and if \texttt{CNTKCTL\_EL1}.EL0PCTEN is 0, \texttt{CNTFRQ}.

<table>
<thead>
<tr>
<th>ELOPCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 accesses to the frequency register and physical counter register are trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented and \texttt{HCR\_EL2}.\{E2H, TGE\} is \{1, 1\}, this control does not cause any instructions to be trapped.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the CNTKCTL\_EL1

When \texttt{HCR\_EL2}.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTKCTL\_EL1 or CNTKCTL\_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, CNTKCTL_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
eslif PSTATE.EL == EL1 then
  return CNTKCTL_EL1;
eslif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return CNTHCTL_EL2;
  else
    return CNTKCTL_EL1;
eslif PSTATE.EL == EL3 then
  return CNTKCTL_EL1;

MSR CNTKCTL_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
eslif PSTATE.EL == EL1 then
  CNTKTL_EL1 = X[t];
eslif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    CNTHCTL_EL2 = X[t];
  else
    CNTKCTL_EL1 = X[t];
eslif PSTATE.EL == EL3 then
  CNTKCTL_EL1 = X[t];

MRS <Xt>, CNTKCTL_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
eslif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
eslif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CNTKCTL_EL1;
  else
    UNDEFINED;
eslif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CNTKCTL_EL1;
  else
    UNDEFINED;

MSR CNTKCTL_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

CNTKCTL_EL1, Counter-timer Kernel Control register
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTKCTL_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTKCTL_EL1 = X[t];
    else
        UNDEFINED;
The CNTP_CTL_EL0 characteristics are:

**Purpose**

Control register for the EL1 physical timer.

**Configuration**

AArch64 System register CNTP_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_CTL[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CNTP_CTL_EL0 is a 64-bit register.

**Field descriptions**

The CNTP_CTL_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>ISTATUS</td>
</tr>
<tr>
<td>30</td>
<td>IMASK</td>
</tr>
<tr>
<td>29</td>
<td>ENABLE</td>
</tr>
</tbody>
</table>

**Bits [63:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is **UNKNOWN**.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:
### IMASK

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

### ENABLE, bit [0]

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from `CNTP_TVAL_EL0` continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the CNTP_CTL_EL0

When `HCR_EL2.E2H` is 1, without explicit synchronization, access from EL3 using the mnemonic `CNTP_CTL_EL0` or `CNTP_CTL_EL02` are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CNTP_CTL_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    end
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTKCTL_EL2.EL1PCEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTKCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHPS_CTL_EL2;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHP_CTL_EL2;
else
    return CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTKCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHPS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTP_CTL_EL0;

MSR CNTP_CTL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  endif
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
  CNTHPS_CTL_EL2 = X[t];
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
  CNTHP_CTL_EL2 = X[t];
else
  CNTP_CTL_EL0 = X[t];
endif
else PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x180] = X[t];
  else
    CNTP_CTL_EL0 = X[t];
  endif
else PSTATE.EL == EL2 then
  if HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '0' then
    CNTHPS_CTL_EL2 = X[t];
  else HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '1' then
    CNTHP_CTL_EL2 = X[t];
  else
    CNTP_CTL_EL0 = X[t];
  endif
else PSTATE.EL == EL3 then
  CNTP_CTL_EL0 = X[t];
endif

MRS <Xt>, CNTP_CTL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x180];
    elif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CTL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CTL_EL0;
    else
        UNDEFINED;
endif

MSR CNTP_CTL_EL02, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x180] = X[t];
    elsif HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_CTL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_CTL_EL0 = X[t];
    else
        UNDEFINED;
endif
The CNTP_CVAL_EL0 characteristics are:

**Purpose**

Holds the compare value for the EL1 physical timer.

**Configuration**

AArch64 System register CNTP_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTP_CVAL[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTP_CVAL_EL0 is a 64-bit register.

**Field descriptions**

The CNTP_CVAL_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**CompareValue, bits [63:0]**

Holds the EL1 physical timer CompareValue.

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTP_CVAL_EL0**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_CVAL_EL0 or CNTP_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
      return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
      return CNTHP_CVAL_EL2;
    else
      return CNTP_CVAL_EL0;
  endif
endif

if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x178];
  else
    return CNTP_CVAL_EL0;
  endif
endif

if PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTHPS_CVAL_EL2;
  elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    return CNTHP_CVAL_EL2;
  else
    return CNTP_CVAL_EL0;
  endif
endif

if PSTATE.EL == EL3 then
  return CNTP_CVAL_EL0;
endif

MSR CNTP_CVAL_EL0, <Xt>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTKCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
            CNTHPS_CVAL_EL2 = X[t];
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
            CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
    endif
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTKCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.NS == '0' then
            NVMem[0x178] = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHPS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t];
end if

MRS <Xt>, CNTP_CVAL_EL02

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x178];
        endif
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;
    endif
endif

MSR CNTP_CVAL_EL02, < Xt >

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x178];
        endif
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;
    endif
endif
The CNTP_TVAL_EL0 characteristics are:

**Purpose**

Holds the timer value for the EL1 physical timer.

**Configuration**

AArch64 System register CNTP_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTP_TVAL[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CNTP_TVAL_EL0 is a 64-bit register.

**Field descriptions**

The CNTP_TVAL_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**RES0**

TimerValue

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL_EL0.ENABLE is 0, the value returned is **UNKNOWN**.
- If CNTP_CTL_EL0.ENABLE is 1, the value returned is (CNTP_CVAL_EL0 - CNTPCT_EL0).

On a write of this register, CNTP_CVAL_EL0 is set to (CNTPCT_EL0 + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CNTP_CVAL_EL0) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL_EL0.ISTATUS is set to 1.
- If CNTP_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTP_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally **UNKNOWN** value.
## Accessing the CNTP_TVAL_EL0

When \texttt{HCR_EL2.E2H} is 1, without explicit synchronization, access from EL3 using the mnemonic \texttt{CNTP_TVAL_EL0} or \texttt{CNTP_TVAL_EL02} are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CNTP_TVAL_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if \texttt{PSTATE.EL == EL0} then
if \texttt{!ELUsingAArch32(EL1) && \texttt{HCR_EL2.<E2H,TGE> == '11'} && \texttt{CNTKCTL_EL1.EL0PTEN == '0'}} then
if \texttt{EL2Enabled()} && \texttt{!ELUsingAArch32(EL2) && \texttt{HCR_EL2.TGE == '1'}} then
AArch64.SystemAccessTrap(EL2, 0x18);
else
AArch64.SystemAccessTrap(EL1, 0x18);

elsif \texttt{EL2Enabled()} && \texttt{!ELUsingAArch32(EL2) && \texttt{HCR_EL2.<E2H,TGE> == '10'}} then
AArch64.SystemAccessTrap(EL2, 0x18);
else
AArch64.SystemAccessTrap(EL2, 0x18);

elsif \texttt{EL2Enabled()} && \texttt{!ELUsingAArch32(EL2) && \texttt{HCR_EL2.<E2H,TGE> == '11'}} && \texttt{CNTHCTL_EL2.EL1PTEN == '0'} then
return \texttt{CNTHPS_TVAL_EL2};
else
return \texttt{CNTHP_TVAL_EL2};
endif

elsif \texttt{PSTATE.EL == EL1} then
if \texttt{EL2Enabled()} && \texttt{!ELUsingAArch32(EL2) && \texttt{HCR_EL2.E2H == '0'} && \texttt{CNTHCTL_EL2.EL1PCEN == '0'}} then
AArch64.SystemAccessTrap(EL2, 0x18);
else
AArch64.SystemAccessTrap(EL2, 0x18);

elsif \texttt{EL2Enabled()} && \texttt{!ELUsingAArch32(EL2) && \texttt{HCR_EL2.E2H == '1'}} && \texttt{CNTHCTL_EL2.EL1PTEN == '0'} then
return \texttt{CNTHPS_TVAL_EL2};
else
return \texttt{CNTHP_TVAL_EL2};
endif

elsif \texttt{PSTATE.EL == EL2} then
if \texttt{HCR_EL2.E2H == '1'} && \texttt{SCR_EL3.NS == '0'} then
return \texttt{CNTHPS_TVAL_EL2};
else
return \texttt{CNTHP_TVAL_EL2};
else
return \texttt{CNTP_TVAL_EL0};
endif

elsif \texttt{PSTATE.EL == EL3} then
return \texttt{CNTP_TVAL_EL0};
else
return \texttt{CNTP_TVAL_EL0};
endif

**MSR CNTP_TVAL_EL0, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.TGE == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL0PTEN == '0' then
    AAerch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
    endif
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  endif
else
  AArch64.SystemAccessTrap(EL1, 0x18);
endif

if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    return CNTP_TVAL_EL0;
  else
    UNDEFINED;
  endif
else
  UNDEFINED;
endif

if PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CNTP_TVAL_EL0;
  else
    UNDEFINED;
  endif
else
  UNDEFINED;
endif

if PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CNTP_TVAL_EL0;
  else
    UNDEFINED;
  endif
else
  UNDEFINED;
endif

MRS <Xt>, CNTP_TVAL_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      UNDEFINED;
    endif
  else
    UNDEFINED;
  endif
else
  UNDEFINED;
else
  UNDEFINED;
endif

if PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CNTP_TVAL_EL0;
  else
    UNDEFINED;
  endif
else
  UNDEFINED;
endif

Page 224
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_TVAL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTP_TVAL_EL0 = X[t];
    else
        UNDEFINED;

---

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
CNTPCT_EL0, Counter-timer Physical Count register

The CNTPCT_EL0 characteristics are:

**Purpose**

Holds the 64-bit physical count value.

**Configuration**

AArch64 System register CNTPCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTPCT[63:0].

All reads to the CNTPCT_EL0 occur in program order relative to reads to CNTPCTSS_EL0 or CNTPCT_EL0.

**Attributes**

CNTPCT_EL0 is a 64-bit register.

**Field descriptions**

The CNTPCT_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Physical count value</td>
</tr>
<tr>
<td>62</td>
<td>Physical count value</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Physical count value</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Physical count value.

Reads of CNTPCT_EL0 from EL0 or EL1 return (PCount<63:0> - CNTPOFF_EL2<63:0>) if the access is not trapped, and all of the following are true:

- CNTHCTL_EL2.ECV is 1.
- HCR_EL2.{E2H, TGE} is not {1, 1}.

Where PCount<63:0> is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

**Accessing the CNTPCT_EL0**

Accesses to this register use the following encodings:

MRS <Xt>, CNTPCT_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
    CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' &&
    CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
    CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTPCT_EL0;
    endif
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTPCT_EL0;
    endif
elsif PSTATE.EL == EL2 then
    return CNTPCT_EL0;
elsif PSTATE.EL == EL3 then
    return CNTPCT_EL0;

CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Count register

The CNTPCTSS_EL0 characteristics are:

**Purpose**

Holds the self-synchronized view of the 64-bit physical count value.

**Configuration**

AArch64 System register CNTPCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTPCTSS[63:0].

This register is present only when ARMv8.6-ECV is implemented. Otherwise, direct accesses to CNTPCTSS_EL0 are UNDEFINED.

All reads to the CNTPCTSS_EL0 occur in program order relative to reads to CNTPCT_EL0 or CNTPCTSS_EL0.

This register is a self-synchronised view of the CNTPCT_EL0 counter, and cannot be read speculatively.

**Attributes**

CNTPCTSS_EL0 is a 64-bit register.

**Field descriptions**

The CNTPCTSS_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>Self-synchronized physical count value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-synchronized physical count value</td>
</tr>
</tbody>
</table>

**Accessing the CNTPCTSS_EL0**

Accesses to this register use the following encodings:

MRS <Xt>, CNTPCTSS_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b110</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
        CNTHCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
            CNTHCTL_EL2.EL1PCTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    endif
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTPCTSS_EL0;
    endif
endif

if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTPCTSS_EL0;
    endif
else
    if PSTATE.EL == EL2 then
        return CNTPCTSS_EL0;
    elseif PSTATE.EL == EL3 then
        return CNTPCTSS_EL0;
    endif
endif
CNTPOFF_EL2, Counter-timer Physical Offset register

The CNTPOFF_EL2 characteristics are:

Purpose

Holds the 64-bit physical offset. This is the offset for the AArch64 physical timers and counters when Enhanced Counter Virtualization is enabled.

Configuration

This register is present only when ARMv8.6-ECV is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are UNDEFINED.

The offsetting of the timers and counters based on EL2 using AArch64 apply at:

- EL1 when EL1 is using AArch64 or AArch32.
- EL0 when EL0 is using AArch64 or AArch32.

When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:

- CNTHCTL_EL2. ECV is 0.
- SCR_EL3. ECVEN is 0.
- HCR_EL2. {E2H, TGE} is {1, 1}.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

CNTPOFF_EL2 is a 64-bit register.

Field descriptions

The CNTPOFF_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Physical offset, bits [63:0]

Physical offset.

This field resets to an architecturally UNKNOWN value.

Accessing the CNTPOFF_EL2

Accesses to this register use the following encodings:

MRS <Xt>, CNTPOFF_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x1A8];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ECVEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    CNTPOFF_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  CNTPOFF_EL2 = X[t];

MSR CNTPOFF_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x1A8] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ECVEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    CNTPOFF_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  CNTPOFF_EL2 = X[t];
CNTPS_CTL_EL1, Counter-timer Physical Secure Timer Control register

The CNTPS_CTL_EL1 characteristics are:

**Purpose**

Control register for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTPS_CTL_EL1 is a 64-bit register.

**Field descriptions**

The CNTPS_CTL_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>ISTATUS</td>
<td>The status of the timer. This bit indicates whether the timer condition is met:</td>
</tr>
<tr>
<td>IMASK</td>
<td>Timer interrupt mask bit. Permitted values are:</td>
</tr>
<tr>
<td>ENABLE</td>
<td></td>
</tr>
</tbody>
</table>

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>
For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTPS_TVAL_EL1** continues to count down.

---

**Note**

Disabling the output signal might be a power-saving option.

---

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTPS_CTL_EL1**

Accesses to this register use the following encodings:

```plaintext
MRS <Xt>, CNTPS_CTL_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b111</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CNTPS_CTL_EL1;
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
        return CNTPS_CTL_EL1;
    MSR CNTPS_CTL_EL1, <Xt>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b111</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EI == EL0 then
    UNDEFINED;
elsi if PSTATE.EI == EL1 then
    if HaveEl(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CTL_EL1 = XI[t];
    else
        UNDEFINED;
    elsif PSTATE.EI == EL2 then
        UNDEFINED;
    elsif PSTATE.EI == EL3 then
        CNTPS_CTL_EL1 = XI[t];

The CNTPS_CVAL_EL1 characteristics are:

**Purpose**

Holds the compare value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTPS_CVAL_EL1 is a 64-bit register.

**Field descriptions**

The CNTPS_CVAL_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|-- |--|
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CNTPS_CVAL_EL1;
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return CNTPS_CVAL_EL1;

MSR CNTPS_CVAL_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b111</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CVAL_EL1 = X[t];
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CVAL_EL1 = X[t];
CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue register

The CNTPS_TVAL_EL1 characteristics are:

**Purpose**

Holds the timer value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CNTPS_TVAL_EL1 is a 64-bit register.

**Field descriptions**

The CNTPS_TVAL_EL1 bit assignments are:

```
 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47
66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36
55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25
44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14
33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the secure physical timer.

On a read of this register:

- If **CNTPS_CTL_EL1**.ENABLE is 0, the value returned is **UNKNOWN**.
- If **CNTPS_CTL_EL1**.ENABLE is 1, the value returned is (**CNTPS_CVAL_EL1** - **CNTPCT_EL0**).

On a write of this register, **CNTPS_CVAL_EL1** is set to (**CNTPCT_EL0** + TimerValue), where TimerValue is treated as a signed 32-bit integer:

When **CNTPS_CTL_EL1**.ENABLE is 1, the timer condition is met when (**CNTPCT_EL0** - **CNTPS_CVAL_EL1**) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- **CNTPS_CTL_EL1**.ISTATUS is set to 1.
- If **CNTPS_CTL_EL1**.IMASK is 0, an interrupt is generated.

When **CNTPS_CTL_EL1**.ENABLE is 0, the timer condition is not met, but **CNTPCT_EL0** continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally **UNKNOWN** value.
Accessing the CNTPS_TVAL_EL1

Accesses to this register use the following encodings:

MRS <Xt>, CNTPS_TVAL_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if HaveEL(EL3) && SCR_EL3.NS == '0' then
      if SCR_EL3.EEL2 == '1' then
         UNDEFINED;
      elsif SCR_EL3.ST == '0' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         return CNTPS_TVAL_EL1;
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   UNDEFINED;
elsif PSTATE.EL == EL3 then
   return CNTPS_TVAL_EL1;

MSR CNTPS_TVAL_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if HaveEL(EL3) && SCR_EL3.NS == '0' then
      if SCR_EL3.EEL2 == '1' then
         UNDEFINED;
      elsif SCR_EL3.ST == '0' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         CNTPS_TVAL_EL1 = X[t];
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   UNDEFINED;
elsif PSTATE.EL == EL3 then
   CNTPS_TVAL_EL1 = X[t];
The CNTV_CTL_EL0 characteristics are:

**Purpose**
Control register for the virtual timer.

**Configuration**
AArch64 System register CNTV_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_CTL[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**
CNTV_CTL_EL0 is a 64-bit register.

**Field descriptions**
The CNTV_CTL_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>ISTATUS</td>
<td>The status of the timer. This bit indicates whether the timer condition is met:</td>
</tr>
<tr>
<td>61</td>
<td>IMASK</td>
<td>Timer interrupt mask bit. Permitted values are:</td>
</tr>
<tr>
<td>60</td>
<td>ENABLE</td>
<td>When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted. When the value of the ENABLE bit is 0, the ISTATUS field is <strong>UNKNOWN</strong>. For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6. This bit is read-only.</td>
</tr>
</tbody>
</table>
Meaning

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally UNKNOWN value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTV_TVAL_EL0 continues to count down.

---

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTV_CTL_EL0**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_CTL_EL0 or CNTV_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CNTV_CTL_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_CTL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elseif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VTV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return NVMem[0x170];
    else
        return CNTV_CTL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_CTL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTV_CTL_EL0;

MSR CNTV_CTL_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTV_CTL_EL2 = X[t];
      else
        CNTV_CTL_EL0 = X[t];
      end
    elsif PSTATE_EL == EL1 then
      if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL2.EL1VTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTV_CTL_EL2 = X[t];
      else
        return NVMem[0x170];
      end
    elsif PSTATE_EL == EL2 then
      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTKCTL_EL2.EL1NVVCT == '1' then
          AArch64.SystemAccessTrap(EL2, 0x18);
        else
          return CNTV_CTL_EL0;
        end
      elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x170] = X[t];
      else
        UNDEFINED;
      end
    elsif PSTATE_EL == EL1 then
      if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTTHVS_CTL_EL2 = X[t];
      elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTTHV_CTL_EL2 = X[t];
      else
        CNTV_CTL_EL0 = X[t];
      end
    elsif PSTATE_EL == EL2 then
      if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTTHVS_CTL_EL2 = X[t];
      elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTTHV_CTL_EL2 = X[t];
      else
        CNTV_CTL_EL0 = X[t];
      end
    elsif PSTATE_EL == EL3 then
      if EL2Enabled() && HCR_EL2.<E2H,TGE> == '1' then
        return CNTV_CTL_EL0;
      else
        UNDEFINED;
      end
    end
  end
end

MRS <Xt>, CNTV_CTL_EL02

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE_EL == EL0 then
  UNDEFINED;
elsif PSTATE_EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTKCTL_EL2.EL1NVVCT == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return NVMem[0x170];
    end
  elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
    return CNTV_CTL_EL0;
  else
    UNDEFINED;
  end
elsif PSTATE_EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CNTV_CTL_EL0;
  else
    UNDEFINED;
  end
elsif PSTATE_EL == EL3 then
  if EL2Enabled() && HCR_EL2.<E2H,TGE> == '1' then
    return CNTV_CTL_EL0;
  else
    UNDEFINED;
end
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x170] = X[t];
        end if
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end if
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTV_CTL_EL0 = X[t];
    else
        UNDEFINED;
    end if
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        CNTV_CTL_EL0 = X[t];
    else
        UNDEFINED;
    end if
end if
The CNTV_CVAL_EL0 characteristics are:

**Purpose**

Holds the compare value for the virtual timer.

**Configuration**

AArch64 System register CNTV_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTV_CVAL[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTV_CVAL_EL0 is a 64-bit register.

**Field descriptions**

The CNTV_CVAL_EL0 bit assignments are:

| Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|     |     |     |     |     |     | CompareValue | CompareValue |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**CompareValue, bits [63:0]**

Holds the EL1 virtual timer CompareValue.

When CNTV_CTL_EL0.ENABLE is 1, the timer condition is met when (CNTVCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTV_CTL_EL0.ISTATUS is set to 1.
- If CNTV_CTL_EL0.IMASK is 0, an interrupt is generated.

When CNTV_CTL_EL0.ENABLE is 0, the timer condition is not met, but CNTVCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTV_CVAL_EL0**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_CVAL_EL0 or CNTV_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
        == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
        == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1TVT
        == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
        then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
        then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
    end
else if PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1TVT
        == '1' then
        return NVMem[0x168];
    else
        return CNTV_CVAL_EL0;
    end
else if PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        return CNTHVS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
    end
else if PSTATE.EL == EL3 then
    return CNTV_CVAL_EL0;

MSR CNTV_CVAL_EL0, <Xt>
if PSTATE_EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    end
else
    AArch64.SystemAccessTrap(EL2, 0x18);
end

MRS <Xt>, CNTV_CVAL_EL02

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && !(HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL1.EL1VTEN == '1' then
            return NVMem[0x168];
        else
            UNDEFINED;
        end
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end
elsif PSTATE_EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTV_CVAL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
    end
else
    CNTV_CVAL_EL0 = X[t];
end
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVVCT == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      NVMem[0x168] = X[t];
    end
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CNTV_CVAL_EL0 = X[t];
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CNTV_CVAL_EL0 = X[t];
  else
    UNDEFINED;
end

---

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The CNTV_TVAL_EL0 characteristics are:

**Purpose**

Holds the timer value for the EL1 virtual timer.

**Configuration**

AArch64 System register CNTV_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register CNTV_TVAL[31:0].

RW fields in this register reset to architecturally unknown values.

**Attributes**

CNTV_TVAL_EL0 is a 64-bit register.

**Field descriptions**

The CNTV_TVAL_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |
| TimerValue |

**Bits [63:32]**

Reserved, RES0.

**TimerValue, bits [31:0]**

The TimerValue view of the EL1 virtual timer.

On a read of this register:

- If `CNTV_CTL_EL0.ENABLE` is 0, the value returned is unknown.
- If `CNTV_CTL_EL0.ENABLE` is 1, the value returned is `(CNTV_CVAL_EL0 - CNTVCT_EL0)`.

On a write of this register, `CNTV_CVAL_EL0` is set to `(CNTVCT_EL0 + TimerValue)`, where TimerValue is treated as a signed 32-bit integer.

When `CNTV_CTL_EL0.ENABLE` is 1, the timer condition is met when `(CNTVCT_EL0 - CNTV_CVAL_EL0)` is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- `CNTV_CTL_EL0.ISTATUS` is set to 1.
- If `CNTV_CTL_EL0.IMASK` is 0, an interrupt is generated.

When `CNTV_CTL_EL0.ENABLE` is 0, the timer condition is not met, but `CNTVCT_EL0` continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally unknown value.
Accessing the CNTV_TVAL_EL0

When \(HCR\text{.}E2H\) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_TVAL_EL0 or CNTV_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

\[
\begin{array}{c|c|c|c|c}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b11 & 0b011 & 0b1110 & 0b0011 & 0b000 \\
\end{array}
\]

if \(PSTATE\text{.}EL == EL0\) then
  if \(!ELUsingAArch32\text{(EL1)}\) && \(!(EL2\text{Enabled()}\) && \(HCR\text{.}EL2.<E2H,\text{TGE}> == '11'\) \&\& \(CNTKCTL\text{.}EL1.EL0VTEN == '0'\) then
    if \(EL2\text{Enabled()}\) \&\& \(!ELUsingAArch32\text{(EL2)}\) \&\& \(HCR\text{.}EL2.\text{TGE} == '1'\) then
      AArch64\text{.}SystemAccessTrap\text{(EL2, 0x18)};
    else
      \text{elseif} \(EL2\text{Enabled()}\) \&\& \(!ELUsingAArch32\text{(EL2)}\) \&\& \(HCR\text{.}EL2.<E2H,\text{TGE}> == '11'\) \&\& \(CNTHCTL\text{.}EL2.\text{EL0VTEN} == '0'\) then
        AArch64\text{.}SystemAccessTrap\text{(EL2, 0x18)};
    \text{elseif} \(HCR\text{.}EL2.\text{E2H} == '1'\) \&\& \(SCR\text{.}EL3.NS == '0'\) then
      return \(CNTV\text{.}TVAL\_EL2\);
    \text{elseif} \(HCR\text{.}EL2.\text{E2H} == '1'\) \&\& \(SCR\text{.}EL3.NS == '1'\) then
      return \(CNTV\text{.}TVAL\_EL2\);
  \text{elseif} \(PSTATE\text{.}EL == EL1\) then
    if \(!ELUsingAArch32\text{(EL1)}\) && \(!(EL2\text{Enabled()}\) && \(HCR\text{.}EL2.<E2H,\text{TGE}> == '11'\) \&\& \(CNTKCTL\text{.}EL1.EL1VTEN == '1'\) then
      AArch64\text{.}SystemAccessTrap\text{(EL2, 0x18)};
    else
      return \(CNTV\text{.}TVAL\_EL0\);
  \text{elseif} \(PSTATE\text{.}EL == EL2\) then
    if \(HCR\text{.}EL2.\text{E2H} == '1'\) \&\& \(SCR\text{.}EL3.NS == '0'\) then
      return \(CNTV\text{.}TVAL\_EL2\);
    \text{elseif} \(HCR\text{.}EL2.\text{E2H} == '1'\) \&\& \(SCR\text{.}EL3.NS == '1'\) then
      return \(CNTV\text{.}TVAL\_EL2\);
  \text{elseif} \(PSTATE\text{.}EL == EL3\) then
    return \(CNTV\text{.}TVAL\_EL0\);
\]

MSR CNTV_TVAL_EL0, \(<Xt>\)
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    end if
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    end if
elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VT == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = X[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_TVAL_EL2 = X[t];
else
    CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTHCTL_EL2.EL1VT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_TVAL_EL2 = X[t];
else
    CNTV_TVAL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_TVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_TVAL_EL0;
    else
        UNDEFINED;
end if

MRS <Xt>, CNTV_TVAL_EL02

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_TVAL_EL0;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return CNTV_TVAL_EL0;
    else
        UNDEFINED;
end if

MSR CNTV_TVAL_EL02, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x10);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CNTV_TVAL_EL0 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CNTV_TVAL_EL0 = X[t];
  else
    UNDEFINED;
CNTVCT_EL0, Counter-timer Virtual Count register

The CNTVCT_EL0 characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value minus the virtual offset visible in CNTVOFF_EL2.

Configuration

AArch64 System register CNTVCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCT[63:0].

The value of this register is the same as the value of CNTPCT_EL0 in the following conditions:

- When EL2 is not implemented.
- When EL2 is implemented, HCR_EL2.E2H is 1, and this register is read from EL2.
- When EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} is {1, 1}, and this register is read from EL0 or EL2.

All reads to the CNTVCT_EL0 occur in program order relative to reads to CNTVCTSS_EL0 or CNTVCT_EL0.

Attributes

CNTVCT_EL0 is a 64-bit register.

Field descriptions

The CNTVCT_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | V    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Virtual count value |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Virtual count value |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

Bits [63:0]

Virtual count value.

Accessing the CNTVCT_EL0

Accesses to this register use the following encodings:

MRS <Xt>, CNTVCT_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') &&
  CNTKCTL_EL1.EL0VCTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
  CNTHCTL_EL2.EL0VCTEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTHCTL_EL2.EL1VCT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return CNTVCT_EL0;
end
elsif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTHCTL_EL2.EL1VCT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return CNTVCT_EL0;
end
elsif PSTATE.EL == EL2 then
  return CNTVCT_EL0;
elsif PSTATE.EL == EL3 then
  return CNTVCT_EL0;
CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count register

The CNTVCTSS_EL0 characteristics are:

**Purpose**

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value visible in CNTPCT_EL0 minus the virtual offset visible in CNTVOFF_EL2.

**Configuration**

AArch64 System register CNTVCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register CNTVCTSS[63:0].

This register is present only when ARMv8.6-ECV is implemented. Otherwise, direct accesses to CNTVCTSS_EL0 are UNDEFINED.

All reads to the CNTVCTSS_EL0 occur in program order relative to reads to CNTPCT_EL0 or CNTVCTSS_EL0.

This register is a self-synchronised view of the CNTPCT_EL0 counter, and cannot be read speculatively.

**Attributes**

CNTVCTSS_EL0 is a 64-bit register.

**Field descriptions**

The CNTVCTSS_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Self-synchronized virtual count value.

**Accessing the CNTVCTSS_EL0**

Accesses to this register use the following encodings:

MRS <Xt>, CNTVCTSS_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
        CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
        CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
        CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTVCTSS_EL0;
    elsif PSTATE.EL == EL1 then
        if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
            CNTHCTL_EL2.EL1TVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return CNTVCTSS_EL0;
        elsif PSTATE.EL == EL2 then
            return CNTVCTSS_EL0;
        elsif PSTATE.EL == EL3 then
            return CNTVCTSS_EL0;
        end
    end
CNTVOFF_EL2, Counter-timer Virtual Offset register

The CNTVOFF_EL2 characteristics are:

**Purpose**

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in CNTPCT_EL0 and the virtual count value visible in CNTVCT_EL0.

**Configuration**

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register CNTVOFF[63:0].

If EL2 is not implemented, this register is RES0 from EL3 and the virtual counter uses a fixed virtual offset of zero.

**Note**

When EL2 is implemented and enabled in the current Security state, and is using AArch64, the virtual counter uses a fixed virtual offset of zero in the following situations:

- HCR_EL2.E2H is 1, and CNTVCT_EL0 is read from EL2.
- HCR_EL2.{E2H, TGE} is {1, 1}, and either:
  - CNTVCT_EL0 is read from EL0 or EL2.
  - CNTVCT is read from EL0.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTVOFF_EL2 is a 64-bit register.

**Field descriptions**

The CNTVOFF_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual offset</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Virtual offset</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>Virtual offset</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTVOFF_EL2**

Accesses to this register use the following encodings:
MRS <Xt>, CNTVOFF_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x060];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    return CNTVOFF_EL2;
elsif PSTATE.EL == EL3 then
    return CNTVOFF_EL2;
endif

MSR CNTVOFF_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x060] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    CNTVOFF_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTVOFF_EL2 = X[t];
endif
The CONTEXTIDR_EL1 characteristics are:

**Purpose**

Identifies the current Process Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

This register is used when ARMv8.1-VHE is not implemented, or when ARMv8.1-VHE is implemented and HCR_EL2.E2H is set to 0.

**Note**

When ARMv8.1-VHE is implemented and HCR_EL2.E2H is set to 1, CONTEXTIDR_EL2 is used.

**Configuration**

AArch64 System register CONTEXTIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CONTEXTIDR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CONTEXTIDR_EL1 is a 64-bit register.

**Field descriptions**

The CONTEXTIDR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | PROCID |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**PROCID, bits [31:0]**

Process Identifier. This field must be programmed with a unique value that identifies the current process.

**Note**

In AArch32 state, when TTBCR.EAE is set to 0, CONTEXTIDR.ASID holds the ASID.
In AArch64 state, CONTEXTIDR_EL1 is independent of the ASID, and for the
EL1&0 translation regime either TTBR0_EL1 or TTBR1_EL1 holds the ASID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CONTEXTIDR_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CONTEXTIDR_EL1 or CONTEXTIDR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CONTEXTIDR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elseslf PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elseslf EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.CONTEXTIDR_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
elseslf EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x108];
    else
        return CONTEXTIDR_EL1;
elseslf PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL2;
elseslf PSTATE.EL == EL3 then
        return CONTEXTIDR_EL1;

**MSR CONTEXTIDR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elseslf PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elseslf EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.CONTEXTIDR_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
elseslf EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x108] = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elseslf PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL2 = X[t];
elseslf PSTATE.EL == EL3 then
        CONTEXTIDR_EL1 = X[t];
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x108];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CONTEXTIDR_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CONTEXTIDR_EL1;
  else
    UNDEFINED;

MSR CONTEXTIDR_EL12, <Xt>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x108] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CONTEXTIDR_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CONTEXTIDR_EL1 = X[t];
  else
    UNDEFINED;
CONTEXTIDR_EL2, Context ID Register (EL2)

The CONTEXTIDR_EL2 characteristics are:

**Purpose**

When HCR_EL2.E2H is set to 1, identifies the current Process Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

**Note**

When HCR_EL2.E2H is 0, CONTEXTIDR_EL2 replaces CONTEXTIDR_EL1 where CONTEXTIDR_EL1 would usually be used.

**Configuration**

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CONTEXTIDR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CONTEXTIDR_EL2 is a 64-bit register.

**Field descriptions**

The CONTEXTIDR_EL2 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| PROCID |

**Bits [63:32]**

Reserved, RES0.

**PROCID, bits [31:0]**

Process Identifier. This field must be programmed with a unique value that identifies the current process.

**Note**

In AArch32 state, when TTBCR.EAE is set to 0, CONTEXTIDR ASID holds the ASID.
In AArch64 state, CONTEXTIDR_EL2 is independent of the ASID, and for the EL2&0 translation regime either TTBR0_EL2 or TTBR1_EL2 holds the ASID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CONTEXTIDR_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic CONTEXTIDR_EL2 or CONTEXTIDR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, CONTEXTIDR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CONTEXTIDR_EL2;
elsif PSTATE.EL == EL3 then
    return CONTEXTIDR_EL2;
```

**MSR CONTEXTIDR_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CONTEXTIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL2 = X[t];
```

**MRS <Xt>, CONTEXTIDR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.CONTEXTIDR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x108];
  else
    return CONTEXTIDR_EL1;
  end;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return CONTEXTIDR_EL2;
  else
    return CONTEXTIDR_EL1;
  end;
elsif PSTATE.EL == EL3 then
  return CONTEXTIDR_EL1;
endif;

MSR CONTEXTIDR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CONTEXTIDR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x108] = X[t];
  else
    CONTEXTIDR_EL1 = X[t];
  end;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    CONTEXTIDR_EL2 = X[t];
  else
    CONTEXTIDR_EL1 = X[t];
  end;
elsif PSTATE.EL == EL3 then
  CONTEXTIDR_EL1 = X[t];
The CPACR_EL1 characteristics are:

**Purpose**

Controls access to trace, SVE, Advanced SIMD and floating-point functionality.

**Configuration**

AArch64 System register CPACR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CPACR[31:0].

When HCR_EL2.{E2H, TGE} == {1, 1}, the fields in this register have no effect on execution at EL0 and EL1. In this case, the controls provided by CPTR_EL2 are used.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CPACR_EL1 is a 64-bit register.

**Field descriptions**

The CPACR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>TTA</td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:29]**

Reserved, RES0.

**TTA, bit [28]**

Traps EL0 and EL1 System register accesses to all implemented trace registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states as follows:

- In AArch64 state, accesses to trace registers are trapped, reported using EC syndrome value 0x18.
- In AArch32 state, MRC and MCR accesses to trace registers are trapped, reported using EC syndrome value 0x05.
- In AArch32 state, MRRC and MCRR accesses to trace registers are trapped, reported using EC syndrome value 0x0C.

<table>
<thead>
<tr>
<th>TTA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control causes EL0 and EL1 System register accesses to all implemented trace registers to be trapped.</td>
</tr>
</tbody>
</table>

**Note**

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and
any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.TTA is 1.

- The Armv8-A architecture does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not implemented, this bit is RE0.

This field resets to an architecturally UNKNOWN value.

**Bits [27:22]**

Reserved, RE0.

**FPEN, bits [21:20]**

Traps EL0 and EL1 accesses to the SVE, Advanced SIMD, and floating-point registers to EL1, reported using EC syndrome value 0x07, or to EL2 reported using EC syndrome value 0x00, when EL2 is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states as follows:

- In AArch64 state, accesses to FPCR, FPSR, any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers. See ‘The SIMD and floating-point registers, V0-V31’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- FPCSR, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers. See ‘Advanced SIMD and floating-point System registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>FPEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>This control causes any instructions at EL0 or EL1 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPACR_EL1.ZEN.</td>
</tr>
<tr>
<td>0b01</td>
<td>This control causes any instructions at EL0 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPACR_EL1.ZEN, but does not cause any instruction at EL1 to be trapped.</td>
</tr>
<tr>
<td>0b10</td>
<td>This control causes any instructions at EL0 or EL1 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPACR_EL1.ZEN.</td>
</tr>
<tr>
<td>0b11</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

Writes to MVFR0, MVFR1 and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

**Note**

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPEXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.FPEN is not 0b11.

This field resets to an architecturally UNKNOWN value.

**Bits [19:18]**

Reserved, RE0.
**ZEN, bits [17:16]**

When SVE is implemented:

Traps SVE instructions and instructions that access SVE System registers at EL0 and EL1 to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.

<table>
<thead>
<tr>
<th>ZEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>This control causes these instructions executed at EL0 or EL1 to be trapped.</td>
</tr>
<tr>
<td>0b01</td>
<td>This control causes these instructions executed at EL0 to be trapped, but does not cause any instruction at EL1 to be trapped.</td>
</tr>
<tr>
<td>0b10</td>
<td>This control causes these instructions executed at EL0 or EL1 to be trapped.</td>
</tr>
<tr>
<td>0b11</td>
<td>This control does not cause any instruction to be trapped.</td>
</tr>
</tbody>
</table>

If SVE is not implemented, this field is RES0.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [15:0]**

Reserved, RES0.

**Accessing the CPACR_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic CPACR_EL1 or CPACR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

\[
\text{MRS} <\text{Xt}>, \text{CPACR}_{-}\text{EL1}\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>opp2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() & & !ELUsingAArch32(EL2) & & (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') & & HFGTR_EL2.CPACR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x100];
    else
        return CPACR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) & & !ELUsingAArch32(EL3) & & CPTR_EL3.TCPAC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return CPTR_EL2;
        else
            return CPACR_EL1;
    elsif PSTATE.EL == EL3 then
        return CPACR_EL1;
    elsif PSTATE.EL == EL3 then
        return CPACR_EL1;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) && SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x100] = X[t];
  else
    CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    CPTR_EL2 = X[t];
  else
    CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  CPACR_EL1 = X[t];

MRS <Xt>, CPACR_EL1

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x100];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return CPACR_EL1;
    else
    return CPACR_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return CPACR_EL1;
  else
    UNDEFINED;

MSR CPACR_EL12, <Xt>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) && SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x100] = X[t];
  else
    CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HCR_EL2.E2H == '1' then
    CPTR_EL2 = X[t];
  else
    CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  CPACR_EL1 = X[t];

MRS <Xt>, CPACR_EL12

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) && SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x100] = X[t];
  else
    CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HCR_EL2.E2H == '1' then
    CPTR_EL2 = X[t];
  else
    CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  CPACR_EL1 = X[t];

MRS <Xt>, CPACR_EL12
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x100] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      CPACR_EL1 = X[t];
    else
      UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    CPACR_EL1 = X[t];
  else
    UNDEFINED;
The CPP RCTX characteristics are:

**Purpose**

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, cache prefetch prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

**Note**

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

**Configuration**

This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to CPP RCTX are UNDEFINED.

**Attributes**

CPP RCTX is a 64-bit System instruction.

**Field descriptions**

The CPP RCTX input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-49</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

Reserved, RES0.
GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

<table>
<thead>
<tr>
<th>GVMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and one of:

- an EL1 context.
- an EL0 context when (HCR_EL2.E2H==0 or HCR_EL2.TGE==0).

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==0 or HCR_EL2.TGE==0) then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==1 and HCR_EL2.TGE==1) then this field is ignored.

Bits [31:27]

Reserved, RES0.

NS, bit [26]

Security State.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state.</td>
</tr>
</tbody>
</table>

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level.

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>EL0</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1</td>
</tr>
<tr>
<td>0b10</td>
<td>EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>EL3</td>
</tr>
</tbody>
</table>

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.
### GASID

<table>
<thead>
<tr>
<th>GASID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0, then this field has an Effective value of 0.

### ASID, bits [15:0]

Only applies for an EL0 context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.

### Executing the CPP RCTX instruction

Accesses to this instruction use the following encodings:

**CPP RCTX, <Xt>**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b111</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif;
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.CPPRCTX == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      else
        CPP_RCTX(X[t]);
      endif;
    else
      CPP_RCTX(X[t]);
    endif;
  else
    CPP_RCTX(X[t]);
  endif;
else
  CPP_RCTX(X[t]);
endif;
```
**CPTR_EL2, Architectural Feature Trap Register (EL2)**

The CPTR_EL2 characteristics are:

### Purpose

**Controls:**

- Trapping to EL2 of access to CPACR, CPACR_EL1, trace functionality, and to SVE, Advanced SIMD and floating-point functionality.
- EL2 access to trace functionality, and to SVE, Advanced SIMD and floating-point functionality.

### Configuration

AArch64 System register CPTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HCPTR[31:0]. If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

CPTR_EL2 is a 64-bit register.

### Field descriptions

The CPTR_EL2 bit assignments are:

**When ARMv8.1-VHE is implemented and HCR_EL2.E2H == 1:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>62</td>
<td>EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TCPAC, bit [31]**

When HCR_EL2.TGE is 0, traps EL1 accesses to CPACR_EL1 reported using EC syndrome value 0x18, and accesses to CPACR reported using EC syndrome value 0x03, to EL2 when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>TCPAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, this control does not cause any instructions to be trapped.

### Note

CPACR_EL1 and CPACR are not accessible at EL0.
When AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - `AMUSERENR_EL0`, `AMCFGR_EL0`, `AMCGCR_EL0`, `AMCNTENCLR0_EL0`, `AMCNTENCLR1_EL0`, `AMCNTENSET0_EL0`, `AMCNTENSET1_EL0`, `AMCR_EL0`, `AMEVCNTR0<n>_EL0`, `AMEVTYPER0<n>_EL0`, and `AMEVTYPER1<n>_EL0`.

- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03:
  - `AMUSERENR`, `AMCFGR`, `AMCGCR`, `AMCNTENCLR0`, `AMCNTENCLR1`, `AMCNTENSET0`, `AMCNTENSET1`, `AMCR`, `AMEVTYPER0<n>`, and `AMEVTYPER1<n>`.

- In AArch32 state, MRRC or MCRR accesses to `AMEVCNTR0<n>` and `AMEVCNTR1<n>`, are trapped to EL2, reported using EC syndrome value 0x04.

<table>
<thead>
<tr>
<th>TAM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses from EL1 and EL0 to Activity Monitor registers are not trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

- Reserved, RES0.

Bit [29]

- Reserved, RES0.

TTA, bit [28]

Traps System register accesses to all implemented trace registers to EL2 when EL2 is enabled in the current Security state, from both Execution states, as follows:

- In AArch64 state, accesses to trace registers with op0=2, op1=1 are trapped to EL2, reported using EC syndrome value 0x18.

- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, are trapped to EL2, reported using EC syndrome value 0x05.

- In AArch32 state, MRRC or MCRR accesses to trace registers with cpnum=14, opc1=1, are trapped to EL2, reported using EC syndrome value 0x0C.

<table>
<thead>
<tr>
<th>TTA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt at EL0, EL1 or EL2, to execute a System register access to an implemented trace register is trapped to EL2 when EL2 is enabled in the current Security state, unless <code>HCR_EL2.TGE</code> is 0 and it is trapped by <code>CPACR.NSTRCDIS</code> or <code>CPACR_EL1.TTA</code>. When <code>HCR_EL2.TGE</code> is 1, any attempt at EL0 or EL2 to execute a System register access to an implemented trace register is trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

Note
The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.

EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not supported, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

### Bits [27:22]

Reserved, RES0.

### FPEN, bits [21:20]

Traps EL0, EL2 and, when HCR_EL2.TGE is 0, EL1 accesses to the SVE, Advanced SIMD and floating-point registers to EL2 when EL2 is enabled in the current Security state, from both Execution states.

<table>
<thead>
<tr>
<th>FPEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>This control causes any instructions at EL0, EL1, or EL2 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, subject to the exception prioritization rules, unless they are trapped by CPTR_EL2.ZEN.</td>
</tr>
<tr>
<td>0b01</td>
<td>When HCR_EL2.TGE is 0, this control does not cause any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes instructions at EL0 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, unless they are trapped by CPTR_EL2.ZEN, but does not cause any instruction at EL2 to be trapped.</td>
</tr>
<tr>
<td>0b10</td>
<td>This control causes any instructions at EL0, EL1, or EL2 that use the registers associated with SVE, Advanced SIMD and floating-point execution to be trapped, subject to the exception prioritization rules, unless they are trapped by CPTR_EL2.ZEN.</td>
</tr>
<tr>
<td>0b11</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

Writes to MVFR0, MVFR1, and MVFR2 from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

---

**Note**

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to FPSID, MVFR0, MVFR1, MVFR2, and FPFXC are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.FPEN is not 0b11.

This field resets to an architecturally UNKNOWN value.

### Bits [19:18]

Reserved, RES0.

### ZEN, bits [17:16]

When SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions or instructions that access SVE System registers to EL2 when EL2 is enabled in the current Security state.
This control causes execution at EL2, EL1, and EL0 of these instructions to be trapped, subject to the exception prioritization rules.

When `HCR_EL2.TGE` is 0, this control does not cause any instruction to be trapped.

When `HCR_EL2.TGE` is 1, this control causes these instructions executed at EL0 to be trapped, but does not cause any instruction at EL2 to be trapped.

This control causes execution at EL2, EL1, and EL0 of these instructions to be trapped, subject to the exception prioritization rules.

This control does not cause any instruction to be trapped.

This field resets to an architecturally `UNKNOWN` value.

**Otherwise:**

Reserved, RES0.

**Bits [15:0]**

Reserved, RES0.

**Otherwise:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1 accesses to <code>CPACR_EL1</code> and <code>CPACR</code> are trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

Traps EL1 accesses to `CPACR_EL1`, reported using EC syndrome value 0x18 and accesses to `CPACR`, reported using EC syndrome value 0x03, to EL2 when EL2 is enabled in the current Security state.

When `HCR_EL2.TGE` is 1, this control does not cause any instructions to be trapped.

**TCPAC, bit [31]**

When AMUv1 is implemented:

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- `TCPAC` and `TAM` are not accessible at EL0.

This field resets to an architecturally `UNKNOWN` value.

**Note**

`CPACR_EL1` and `CPACR` are not accessible at EL0.
• In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTESET0_EL0, AMCNTESET1_EL0, AMCR_EL0, AMEVCNTR0<n>_EL0, AMEVCNTR1<n>_EL0, AMEVTYPER0<n>_EL0, and AMEVTYPER1<n>_EL0.

• In AArch32 state, MCR or MRC accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03:
  - AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTESET0, AMCNTESET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.

• In AArch32 state, MCRR or MRRC accesses to AMEVCNTR0<n> and AMEVCNTR1<n>, are trapped to EL2, reported using EC syndrome value 0x04.

<table>
<thead>
<tr>
<th>TAM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses from EL1 and EL0 to Activity Monitor registers are not trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [29:21]**

Reserved, RES0.

**TTA, bit [20]**

Traps System register accesses to all implemented trace registers to EL2 when EL2 is enabled in the current Security state, from both Execution states as follows:

• In AArch64 state, accesses to trace registers with op0=2, op1=1 are trapped to EL2, reported using EC syndrome value 0x18.

• In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1 are trapped to EL2, reported using EC syndrome value 0x05.

• In AArch32 state, MRRC or MCRR accesses to trace registers with cpnum=14, opc1=1 are trapped to EL2, reported using EC syndrome value 0x0C.

<table>
<thead>
<tr>
<th>TTA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt at EL0, EL1, or EL2, to execute a System register access to an implemented trace register is trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by CPACR.TRCDIS or CPACR.EL1.TTA.</td>
</tr>
</tbody>
</table>

**Note**

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.
- EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not supported, this bit is RES0.
This field resets to an architecturally **UNKNOWN** value.

**Bits [19:14]**

Reserved, **RES0**.

**Bits [13:12]**

Reserved, **RES1**.

**Bit [11]**

Reserved, **RES0**.

**TFP, bit [10]**

Traps accesses to SVE, Advanced SIMD and floating-point functionality to EL2 when EL2 is enabled in the current Security state, from both Execution states, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x07:
  - **FPCR, FPSR, FPEXC32_EL2**, any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers. See 'The SIMD and floating-point registers, V0-V31' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x07:
  - **MVFR0, MVFR1, MVFR2, FPCSR, FPEXC**, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers. See ‘Advanced SIMD and floating-point System registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile. For the purposes of this trap, the architecture defines a VMSR access to **FPSID** from EL1 or higher as an access to a SIMD and floating point register. Otherwise, permitted VMSR accesses to **FPSID** are ignored.

<table>
<thead>
<tr>
<th>TFP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt at EL0, EL1 or EL2, to execute an instruction that uses the registers associated with SVE, Advanced SIMD and floating-point execution is trapped to EL2 when EL2 is enabled in the current Security state, subject to the exception prioritization rules, unless it is trapped by <strong>CPTR_EL2.TZ</strong>.</td>
</tr>
</tbody>
</table>

**Note**

**FPEXC32_EL2** is not accessible from EL0 using AArch64.

**FPSID, MRFR0, MVFR1, and FPEXC** are not accessible from EL0 using AArch32.

This field resets to an architecturally **UNKNOWN** value.

**Bit [9]**

Reserved, **RES1**.

**TZ, bit [8]**

**When SVE is implemented:**

Traps execution at EL2, EL1, or EL0 of SVE instructions and instructions that access SVE System registers to EL2 when EL2 is enabled in the current Security state.
### TZ

<table>
<thead>
<tr>
<th>TZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instruction to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control causes these instructions to be trapped, subject to the exception prioritization rules.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES1.

**Bits [7:0]**

Reserved, RES1.

## Accessing the CPTR_EL2

Accesses to this register use the following encodings:

### MRS <Xt>, CPTR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```plaintext
def if PSTATE.EL == EL0 then
    UNDEFINED;
def elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CPTR_EL2;
    elsif PSTATE.EL == EL3 then
        return CPTR_EL2;
def
def MSR CPTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```plaintext
def if PSTATE.EL == EL0 then
    UNDEFINED;
def elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CPTR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        CPTR_EL2 = X[t];
```
MRS <Xt>, CPACR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
else
  return CPACR_EL1;
endif

else if PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    CPTR_EL2 = X[t];
else
  CPACR_EL1 = X[t];
endif

else if PSTATE.EL == EL3 then
  CPACR_EL1 = X[t];
endif

MSR CPACR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWR_EL2.CPACR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
else
  return CPACR_EL1;
endif

else if PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    CPTR_EL2 = X[t];
else
  CPACR_EL1 = X[t];
endif

else if PSTATE.EL == EL3 then
  CPACR_EL1 = X[t];
endif
The CPTR_EL3 characteristics are:

**Purpose**

Controls trapping to EL3 of access to CPACR_EL1, CPTR_EL2, trace functionality and registers associated with SVE, Advanced SIMD and floating-point execution. Also controls EL3 access to trace functionality and registers associated with SVE, Advanced SIMD and floating-point execution.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to CPTR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CPTR_EL3 is a 64-bit register.

**Field descriptions**

The CPTR_EL3 bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| TCPAC  | TAM    | RES0   | TTA    | RES0   | TFP    | RES0   | EZ     | RES0   | RES0   | RES0   | RES0   | RES0   | RES0   | RES0   | RES0   | RES0   |
| 63     | 62     | 61     | 60     | 59     | 58     | 57     | 56     | 55     | 54     | 53     | 52     | 51     | 50     | 49     | 48     | 47     | 46     | 45     | 44     | 43     | 42     | 41     | 40     | 39     | 38     | 37     | 36     | 35     | 34     | 33     | 32     |

**Bits [63:32]**

Reserved, RES0.

**TCPAC, bit [31]**

Traps all of the following to EL3, from both Security states and both Execution states.

- EL2 accesses to CPTR_EL2, reported using EC syndrome value 0x18, or HCPTR, reported using EC syndrome value 0x03.
- EL2 and EL1 accesses to CPACR_EL1 reported using EC syndrome value 0x18, or CPACR reported using EC syndrome value 0x03.

When CPTR_EL3.TCPAC is:

<table>
<thead>
<tr>
<th>TCPAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 accesses to the CPTR_EL2 or HCPTR, and EL2 and EL1 accesses to the CPACR_EL1 or CPACR, are trapped to EL3, unless they are trapped by CPTR_EL2.TCPAC.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**TAM, bit [30]**

When AMUv1 is implemented:

Trap Activity Monitor access. Traps EL2, EL1 and EL0 accesses to all Activity Monitor registers to EL3.

Accesses to the Activity Monitors registers are trapped as follows:
In AArch64 state, the following registers are trapped to EL3 and reported with EC syndrome value \(0x18\):

- AMUSERENR_EL0, AMCFGR_EL0, AMCGCR_EL0, AMCNTENCLR0_EL0, AMCNTENCLR1_EL0, AMCNTENSET0_EL0, AMCNTENSET1_EL0, AMCR_EL0, AMEVCTR0<n>_EL0, AMEVCTR1<n>_EL0, and AMEVTYPER0<n>_EL0.

In AArch32 state, accesses with MRC or MCR to the following registers reported with EC syndrome value \(0x03\):

- AMUSERENR, AMCFGR, AMCGCR, AMCNTENCLR0, AMCNTENCLR1, AMCNTENSET0, AMCNTENSET1, AMCR, AMEVTYPER0<n>, and AMEVTYPER1<n>.

In AArch32 state, accesses with MRRC or MCRR to the following registers, reported with EC syndrome value \(0x04\):

- AMEVCTR0<n>, AMEVCTR1<n>.

### TAM

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
<tr>
<td><strong>0b1</strong></td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### Otherwise:

Reserved, RES0.

### Bits [29:21]

Reserved, RES0.

### TTA, bit [20]

Traps System register accesses. Accesses to the trace registers, from all Exception levels, both Security states, and both Execution states are trapped to EL3 as follows:

- In AArch64 state, Trace registers with \(op0=2\), \(op1=1\), are trapped to EL3 and reported using EC syndrome value \(0x18\).
- In AArch32 state, accesses using MCR or MRC to the Trace registers with \(cpnum=14\) and \(opc1=1\) are reported using EC syndrome value \(0x05\).
- In AArch32 state, accesses using MCRR or MRRC to the Trace registers with \(cpnum=14\) and \(opc1=1\) are reported using EC syndrome value \(0x0C\).

### TTA

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
<tr>
<td><strong>0b1</strong></td>
</tr>
</tbody>
</table>

If System register access to trace functionality is not supported, this bit is RES0.

### Note

The ETMv4 architecture does not permit EL0 to access the trace registers. If the Armv8-A architecture is implemented with an ETMv4 implementation, EL0 accesses to the trace registers are **UNDEFINED**, and any resulting exception is higher priority than this trap exception.

EL3 does not provide traps on trace register accesses through the Memory-mapped interface.
System register accesses to the trace registers can have side-effects. When a System register access is trapped, no side-effects occur before the exception is taken, see 'Register access instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**Bits [19:11]**

Reserved, RES0.

**TFP, bit [10]**

Traps all accesses to SVE, Advanced SIMD and floating-point functionality, from all Exception levels, both Security states, and both Execution states, to EL3. Defined values are:

This includes the following registers, all reported using EC syndrome value 0x07:

- FPCR, FPSR, FPEXC32_EL2, any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers. See 'The SIMD and floating-point registers, V0-V31' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- MVFR0, MVFR1, MVFR2, FPCSR, FPEXC, and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers. See 'Advanced SIMD and floating-point System registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Permitted VMSR accesses to FPSID are ignored, but for the purposes of this trap the architecture define a VMSR access to the FPSID from EL1 or higher as an access to a SIMD and floating-point register.

<table>
<thead>
<tr>
<th>TFP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt at any Exception level to execute an instruction that uses the registers associated with SVE, Advanced SIMD and floating-point is trapped to EL3, subject to the exception prioritization rules, unless it is trapped by CPTR_EL3.EZ.</td>
</tr>
</tbody>
</table>

**Note**

FPEXC32_EL2 is not accessible from EL0 using AArch64.

FPSID, MRFR0, MVFR1, and FPEXC are not accessible from EL0 using AArch32.

This field resets to an architecturally UNKNOWN value.

**Bit [9]**

Reserved, RES0.

**EZ, bit [8]**

**When SVE is implemented:**

Traps all accesses to SVE functionality and registers from all Exception levels, and both Security states, to EL3.

<table>
<thead>
<tr>
<th>EZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control causes these instructions executed at any Exception level to be trapped, subject to the exception prioritization rules.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instruction to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
Bits [7:0]

Reserved, RES0.

Accessing the CPTR_EL3

Accesses to this register use the following encodings:

MRS <Xt>, CPTR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsiif PSTATE.EL == EL1 then
  UNDEFINED;
elsiif PSTATE.EL == EL2 then
  UNDEFINED;
elsiif PSTATE.EL == EL3 then
  return CPTR_EL3;

MSR CPTR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsiif PSTATE.EL == EL1 then
  UNDEFINED;
elsiif PSTATE.EL == EL2 then
  UNDEFINED;
elsiif PSTATE.EL == EL3 then
  CPTR_EL3 = X[t];
CSSELR_EL1, Cache Size Selection Register

The CSSELR_EL1 characteristics are:

**Purpose**

Selects the current Cache Size ID Register, CSIDR_EL1, by specifying the required cache level and the cache type (either instruction or data cache).

**Configuration**

AArch64 System register CSSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register CSSEL[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CSSELR_EL1 is a 64-bit register.

**Field descriptions**

The CSSELR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>61</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>60</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>59</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>58</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>57</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>56</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>55</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>54</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>53</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>52</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>51</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>50</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>49</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>48</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>47</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>46</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>45</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>44</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>43</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>42</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>41</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>40</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>39</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>38</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>37</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>36</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>35</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>34</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>33</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>32</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>31</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>30</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>29</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>28</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>27</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>26</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>25</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>24</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>23</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>22</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>21</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>20</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>19</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>18</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>17</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>16</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>15</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>14</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>13</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>12</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>11</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>10</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>9</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>8</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>7</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>6</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>5</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>4</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>3</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>2</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>1</td>
<td>TnD, bit [4]</td>
</tr>
<tr>
<td>0</td>
<td>TnD, bit [4]</td>
</tr>
</tbody>
</table>

**Bits [63:5]**

Reserved, RES0.

**TnD, bit [4]**

*When ARMv8.5-MemTag is implemented:*

Allocation Tag not Data bit.

<table>
<thead>
<tr>
<th>TnD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Data, Instruction or Unified cache.</td>
</tr>
<tr>
<td>0b1</td>
<td>Separate Allocation Tag cache.</td>
</tr>
</tbody>
</table>

When CSSELR_EL1.InD == 1, this bit is RES0.

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR_EL1 is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Level, bits [3:1]**

Cache level of required cache.
## Level, Meaning

<table>
<thead>
<tr>
<th>Level</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Level 1 cache.</td>
</tr>
<tr>
<td>0b001</td>
<td>Level 2 cache.</td>
</tr>
<tr>
<td>0b010</td>
<td>Level 3 cache.</td>
</tr>
<tr>
<td>0b011</td>
<td>Level 4 cache.</td>
</tr>
<tr>
<td>0b100</td>
<td>Level 5 cache.</td>
</tr>
<tr>
<td>0b101</td>
<td>Level 6 cache.</td>
</tr>
<tr>
<td>0b110</td>
<td>Level 7 cache.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If CSSEL_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSEL_EL1 is **UNKNOWN**.

This field resets to an architecturally **UNKNOWN** value.

### InD, bit [0]

Instruction not Data bit.

<table>
<thead>
<tr>
<th>InD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Data or unified cache.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction cache.</td>
</tr>
</tbody>
</table>

If CSSEL_EL1.Level is programmed to a cache level that is not implemented, then a read of CSSEL_EL1 is **CONSTRAINED UNPREDICTABLE**, and returns **UNKNOWN** values for CSSEL_EL1.{Level, InD}.

This field resets to an architecturally **UNKNOWN** value.

## Accessing the CSSEL_EL1

Accesses to this register use the following encodings:

### MRS <Xt>, CSSEL_EL1

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b010</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRTR_EL2.CSSEL_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return CSSEL_EL1;
  endif
elsif PSTATE.EL == EL2 then
  return CSSEL_EL1;
else
  return CSSEL_EL1;
end
```

### MSR CSSEL_EL1, <Xt>

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b010</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.CSSELR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    CSSELR_EL1 = X[t];
  end
elsif PSTATE.EL == EL2 then
  CSSELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  CSSELR_EL1 = X[t];
CTR_EL0, Cache Type Register

The CTR_EL0 characteristics are:

**Purpose**

Provides information about the architecture of the caches.

**Configuration**

AArch64 System register CTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register CTR[31:0].

**Attributes**

CTR_EL0 is a 64-bit register.

**Field descriptions**

The CTR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>TminLine</td>
</tr>
<tr>
<td>61</td>
<td>RES0</td>
</tr>
<tr>
<td>60</td>
<td>DIC</td>
</tr>
<tr>
<td>59</td>
<td>IDC</td>
</tr>
<tr>
<td>58</td>
<td>CWG</td>
</tr>
<tr>
<td>57</td>
<td>ERG</td>
</tr>
<tr>
<td>56</td>
<td>DminLine</td>
</tr>
<tr>
<td>55</td>
<td>L1Ip</td>
</tr>
<tr>
<td>54</td>
<td>RES0</td>
</tr>
<tr>
<td>53</td>
<td>IminLine</td>
</tr>
<tr>
<td>52</td>
<td>RESERVED</td>
</tr>
<tr>
<td>51</td>
<td>RESERVED</td>
</tr>
<tr>
<td>50</td>
<td>RESERVED</td>
</tr>
<tr>
<td>49</td>
<td>RESERVED</td>
</tr>
<tr>
<td>48</td>
<td>RESERVED</td>
</tr>
<tr>
<td>47</td>
<td>RESERVED</td>
</tr>
<tr>
<td>46</td>
<td>RESERVED</td>
</tr>
<tr>
<td>45</td>
<td>RESERVED</td>
</tr>
<tr>
<td>44</td>
<td>RESERVED</td>
</tr>
<tr>
<td>43</td>
<td>RESERVED</td>
</tr>
<tr>
<td>42</td>
<td>RESERVED</td>
</tr>
<tr>
<td>41</td>
<td>RESERVED</td>
</tr>
<tr>
<td>40</td>
<td>RESERVED</td>
</tr>
<tr>
<td>39</td>
<td>RESERVED</td>
</tr>
<tr>
<td>38</td>
<td>RESERVED</td>
</tr>
<tr>
<td>37</td>
<td>RESERVED</td>
</tr>
<tr>
<td>36</td>
<td>RESERVED</td>
</tr>
<tr>
<td>35</td>
<td>RESERVED</td>
</tr>
<tr>
<td>34</td>
<td>RESERVED</td>
</tr>
<tr>
<td>33</td>
<td>RESERVED</td>
</tr>
<tr>
<td>32</td>
<td>RESERVED</td>
</tr>
<tr>
<td>31</td>
<td>RESERVED</td>
</tr>
<tr>
<td>30</td>
<td>RESERVED</td>
</tr>
<tr>
<td>29</td>
<td>RESERVED</td>
</tr>
<tr>
<td>28</td>
<td>RESERVED</td>
</tr>
<tr>
<td>27</td>
<td>RESERVED</td>
</tr>
<tr>
<td>26</td>
<td>RESERVED</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
</tr>
<tr>
<td>24</td>
<td>RESERVED</td>
</tr>
<tr>
<td>23</td>
<td>RESERVED</td>
</tr>
<tr>
<td>22</td>
<td>RESERVED</td>
</tr>
<tr>
<td>21</td>
<td>RESERVED</td>
</tr>
<tr>
<td>20</td>
<td>RESERVED</td>
</tr>
<tr>
<td>19</td>
<td>RESERVED</td>
</tr>
<tr>
<td>18</td>
<td>RESERVED</td>
</tr>
<tr>
<td>17</td>
<td>RESERVED</td>
</tr>
<tr>
<td>16</td>
<td>RESERVED</td>
</tr>
<tr>
<td>15</td>
<td>RESERVED</td>
</tr>
<tr>
<td>14</td>
<td>RESERVED</td>
</tr>
<tr>
<td>13</td>
<td>RESERVED</td>
</tr>
<tr>
<td>12</td>
<td>RESERVED</td>
</tr>
<tr>
<td>11</td>
<td>RESERVED</td>
</tr>
<tr>
<td>10</td>
<td>RESERVED</td>
</tr>
<tr>
<td>9</td>
<td>RESERVED</td>
</tr>
<tr>
<td>8</td>
<td>RESERVED</td>
</tr>
<tr>
<td>7</td>
<td>RESERVED</td>
</tr>
<tr>
<td>6</td>
<td>RESERVED</td>
</tr>
<tr>
<td>5</td>
<td>RESERVED</td>
</tr>
<tr>
<td>4</td>
<td>RESERVED</td>
</tr>
<tr>
<td>3</td>
<td>RESERVED</td>
</tr>
<tr>
<td>2</td>
<td>RESERVED</td>
</tr>
<tr>
<td>1</td>
<td>RESERVED</td>
</tr>
<tr>
<td>0</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

**Bits [63:38]**

Reserved, RES0.

**TminLine, bits [37:32]**

When ARMv8.5-MemTag is implemented:

Tag minimum Line. Log2 of the number of words covered by Allocation Tags in the smallest cache line of all caches which can contain Allocation tags that are controlled by the PE.

**Note**

For an implementation with cache lines containing 64 bytes of data and 4 Allocation Tags this will be log2(64/4) = 4.

For an implementation with Allocations Tags in separate cache lines of 128 Allocation Tags per line this will be log2(128*16/4) = 9.

**Otherwise:**

Reserved, RES0.

**Bit [31]**

Reserved, RES1.
**Bit [30]**

Reserved, RES0.

**DIC, bit [29]**

Instruction cache invalidation requirements for instruction to data coherence. The meaning of this bit is:

<table>
<thead>
<tr>
<th>DIC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction cache invalidation to the Point of Unification is required for instruction to data coherence.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction cache cleaning to the Point of Unification is not required for instruction to data coherence.</td>
</tr>
</tbody>
</table>

**IDC, bit [28]**

Data cache clean requirements for instruction to data coherence. The meaning of this bit is:

<table>
<thead>
<tr>
<th>IDC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Data cache clean to the Point of Unification is required for instruction to data coherence, unless CLIDR_EL1.LoC == 0b000 or (CLIDR_EL1.LoUIS == 0b000 &amp;&amp; CLIDR_EL1.LoUU == 0b000).</td>
</tr>
<tr>
<td>0b1</td>
<td>Data cache clean to the Point of Unification is not required for instruction to data coherence.</td>
</tr>
</tbody>
</table>

**CWG, bits [27:24]**

Cache writeback granule. \( \log_2 \) of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified.

A value of 0b0000 indicates that this register does not provide Cache writeback granule information and either:

- The architectural maximum of 512 words (2KB) must be assumed.
- The Cache writeback granule can be determined from maximum cache line size encoded in the Cache Size ID Registers.

Values greater than 0b1001 are reserved.

Arm recommends that an implementation that does not support cache write-back implements this field as 0b0001. This applies, for example, to an implementation that supports only write-through caches.

**ERG, bits [23:20]**

Exclusives reservation granule, and, if TME is implemented, transactional reservation granule. \( \log_2 \) of the number of words of the maximum size of the reservation granule for the Load-Exclusive and Store-Exclusive instructions, and, if TME is implemented, for detecting transactional conflicts.

A value of 0b0000 indicates that this register does not provide granule information and the architectural maximum of 512 words (2KB) must be assumed.

Value 0b0001 and values greater than 0b1001 are reserved.

**DminLine, bits [19:16]**

\( \log_2 \) of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE.

**L1Ip, bits [15:14]**

Level 1 instruction cache policy. Indicates the indexing and tagging policy for the L1 instruction cache. Possible values of this field are:
### L1Ip Meaning

<table>
<thead>
<tr>
<th>L1Ip</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>VMID aware Physical Index, Physical tag (VPIPT)</td>
</tr>
<tr>
<td>0b01</td>
<td>ASID-tagged Virtual Index, Virtual Tag (AIVVT)</td>
</tr>
<tr>
<td>0b10</td>
<td>Virtual Index, Physical Tag (VIPT)</td>
</tr>
<tr>
<td>0b11</td>
<td>Physical Index, Physical Tag (PIPT)</td>
</tr>
</tbody>
</table>

The value 0b01 is reserved in Armv8.

The value 0b00 is permitted only in an implementation that includes ARMv8.2-PIPTV, otherwise the value is reserved.

**Bits [13:4]**

- Reserved, RES0.

**IminLine, bits [3:0]**

$\log_2$ of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE.

### Accessing the CTR_EL0

Accesses to this register use the following encodings:

```plaintext
MRS <Xt>, CTR_EL0
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTL_EL1.UCT == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TID2 == '1' then
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.CTR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  endif
else
  return CTR_EL0;
endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return CTR_EL0;
  endif
elsif PSTATE.EL == EL2 then
  return CTR_EL0;
elsif PSTATE.EL == EL3 then
  return CTR_EL0;
```

**Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.**
CurrentEL, Current Exception Level

The CurrentEL characteristics are:

**Purpose**

Holds the current Exception level.

**Configuration**

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CurrentEL is a 64-bit register.

**Field descriptions**

The CurrentEL bit assignments are:

| Bits [63:4] | Reserved, RES0. |
| EL, bits [3:2] | Current Exception level. Possible values of this field are: |
| EL | Meaning |
| 0b00 | EL0 |
| 0b01 | EL1 |
| 0b10 | EL2 |
| 0b11 | EL3 |

When the **HCR_EL2**.NV bit is 1, EL1 read accesses to the CurrentEL register return the value of 0b10 in this field. This field resets to the highest implemented Exception Level.

**Bits [1:0]**

Reserved, RES0.

**Accessing the CurrentEL**

Accesses to this register use the following encodings:

**MRS <Xt>, CurrentEL**
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() \&\& HCR_EL2.NV == '1' then
        return Zeros(60):'10':Zeros(2);
    else
        return Zeros(60):PSTATE.EL:Zeros(2);
    end if;
elsif PSTATE.EL == EL2 then
    return Zeros(60):PSTATE.EL:Zeros(2);
elsif PSTATE.EL == EL3 then
    return Zeros(60):PSTATE.EL:Zeros(2);
DACR32_EL2, Domain Access Control Register

The DACR32_EL2 characteristics are:

**Purpose**

Allows access to the AArch32 DACR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

**Configuration**

AArch64 System register DACR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register DACR[31:0].

If EL1 does not support AArch32, this register is **UNDEFINED**.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is **RES0**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DACR32_EL2 is a 64-bit register.

**Field descriptions**

The DACR32_EL2 bit assignments are:

```
<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>D15</td>
<td>D14</td>
<td>D13</td>
<td>D12</td>
<td>D11</td>
<td>D10</td>
<td>D9</td>
<td>D8</td>
<td>D7</td>
<td>D6</td>
<td>D5</td>
<td>D4</td>
<td>D3</td>
<td>D2</td>
<td>D1</td>
<td>D0</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Bits [63:32]**

Reserved, RES0.

**D<n>**, bits [2n+1:2n], for n = 0 to 15

Domain n access permission, where n = 0 to 15. Permitted values are:

<table>
<thead>
<tr>
<th>D&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>No access. Any access to the domain generates a Domain fault.</td>
</tr>
<tr>
<td>0b01</td>
<td>Client. Accesses are checked against the permission bits in the translation tables.</td>
</tr>
<tr>
<td>0b11</td>
<td>Manager. Accesses are not checked against the permission bits in the translation tables.</td>
</tr>
</tbody>
</table>

The value 0b10 is reserved.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the DACR32_EL2**

Accesses to this register use the following encodings:
MRS <Xt>, DACR32_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  return DACR32_EL2;
elsif PSTATE.EL == EL3 then
  return DACR32_EL2;

MSR DACR32_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  DACR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  DACR32_EL2 = X[t];
DAIF, Interrupt Mask Bits

The DAIF characteristics are:

**Purpose**

Allows access to the interrupt mask bits.

**Configuration**

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DAIF is a 64-bit register.

**Field descriptions**

The DAIF bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:10]**

Reserved, RES0.

**D, bit [9]**

Process state D mask. The possible values of this bit are:

<table>
<thead>
<tr>
<th>D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint, Breakpoint, and Software Step exceptions targeted at the current Exception level are not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint, Breakpoint, and Software Step exceptions targeted at the current Exception level are masked.</td>
</tr>
</tbody>
</table>

When the target Exception level of the debug exception is higher than the current Exception level, the exception is not masked by this bit.

This field resets to 1.

**A, bit [8]**

SError interrupt mask bit. The possible values of this bit are:

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked.</td>
</tr>
</tbody>
</table>

This field resets to 1.
I, bit [7]

IRQ mask bit. The possible values of this bit are:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked.</td>
</tr>
</tbody>
</table>

This field resets to 1.

F, bit [6]

FIQ mask bit. The possible values of this bit are:

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked.</td>
</tr>
</tbody>
</table>

This field resets to 1.

Bits [5:0]

Reserved, RES0.

Accessing the DAIF

For details on the operation of the MSR (immediate) accessor, see MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

MRS <Xt>, DAIF

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && ((EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') || SCTLR_EL1.UMA == '0') then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
  else
    return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
else
  if PSTATE.EL == EL1 then
    return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
  else if PSTATE.EL == EL2 then
    return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
  else if PSTATE.EL == EL3 then
    return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);

MSR DAIF, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && ((EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') || SCTLR_EL1.UMA == '0') then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if;
    else
        PSTATE.<D,A,I,F> = X[t]<9:6>;
    end if;
elsif PSTATE.EL == EL1 then
    PSTATE.<D,A,I,F> = X[t]<9:6>;
elsif PSTATE.EL == EL2 then
    PSTATE.<D,A,I,F> = X[t]<9:6>;
elsif PSTATE.EL == EL3 then
    PSTATE.<D,A,I,F> = X[t]<9:6>;
end if;

MSR DAIFSet, #imm

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b110</td>
</tr>
</tbody>
</table>

MSR DAIFClr, #imm

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
DBGAUTHSTATUS_EL1, Debug Authentication Status register

The DBGAUTHSTATUS_EL1 characteristics are:

**Purpose**

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

**Configuration**

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGAUTHSTATUS[31:0].

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to External register DBGAUTHSTATUS_EL1[31:0].

**Attributes**

DBGAUTHSTATUS_EL1 is a 64-bit register.

**Field descriptions**

The DBGAUTHSTATUS_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>SNID, bits [7:6]</td>
</tr>
<tr>
<td>61</td>
<td>When ARMv8.4-Debug is implemented:</td>
</tr>
<tr>
<td>60</td>
<td>Secure non-invasive debug.</td>
</tr>
<tr>
<td>59</td>
<td>This field has the same value as DBGAUTHSTATUS_EL1.SID.</td>
</tr>
<tr>
<td>58</td>
<td>Otherwise:</td>
</tr>
<tr>
<td>57</td>
<td>Secure non-invasive debug.</td>
</tr>
<tr>
<td>56</td>
<td>SNID</td>
</tr>
<tr>
<td>55</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1.</td>
</tr>
<tr>
<td>54</td>
<td>Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>53</td>
<td>Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.
**SID, bits [5:4]**

Secure invasive debug.

<table>
<thead>
<tr>
<th>SID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NSNID, bits [3:2]**

When ARMv8.4-Debug is implemented:

Non-secure non-invasive debug.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. EL3 is implemented or the Effective value of SCR_EL3.NS is 1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NSID, bits [1:0]**

Non-secure invasive debug.

<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the DBGAUTHSTATUS_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, DBGAUTHSTATUS_EL1

```
| op0 | op1 | CRn | CRm | op2 |
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.DBGAUTHSTATUS_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGAUTHSTATUS_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGAUTHSTATUS_EL1;
    end
elsif PSTATE.EL == EL3 then
    return DBGAUTHSTATUS_EL1;
The DBGBCR\textsubscript{n} EL1 characteristics are:

**Purpose**

Holds control information for a breakpoint. Forms breakpoint \(n\) together with value register DBGVR\textsubscript{n} EL1.

**Configuration**

AArch64 System register DBGBCR\textsubscript{n} EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBCR\textsubscript{n}[31:0].

AArch64 System register DBGBCR\textsubscript{n} EL1 bits [31:0] are architecturally mapped to External register DBGBCR\textsubscript{n} EL1[31:0].

If breakpoint \(n\) is not implemented then accesses to this register are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGBCR\textsubscript{n} EL1 is a 64-bit register.

**Field descriptions**

The DBGBCR\textsubscript{n} EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BT | LBN | SSC | HMC | RES0 | BAS | RES0 | PMC | E |

**Bits [63:24]**

Reserved, RES0.

**BT, bits [23:20]**

Breakpoint Type. Possible values are:
<table>
<thead>
<tr>
<th>BT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Unlinked instruction address match. DBGBVR&lt;-&gt;_EL1 is the address of an instruction.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As 0b0000, but linked to a Context matching breakpoint.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Unlinked Context ID match. When ARMv8.1-VHE is implemented, EL2 is using AArch64, and the Effective value of HCR_EL2.E2H is 1, if either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR&lt;-&gt;_EL1.ContextID must match the CONTEXTIDR_EL2 value. Otherwise, DBGBVR&lt;-&gt;_EL1.ContextID must match the CONTEXTIDR_EL1 value</td>
</tr>
<tr>
<td>0b0011</td>
<td>As 0b0010, with linking enabled.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Unlinked VMID match. DBGBVR&lt;-&gt;_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As 0b1000, with linking enabled.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Unlinked VMID and Context ID match. DBGBVR&lt;-&gt;_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1, and DBGBVR&lt;-&gt;_EL1.VMID is a VMID compared against VTTBR_EL2.VMID.</td>
</tr>
<tr>
<td>0b0111</td>
<td>As 0b1010, with linking enabled.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Unlinked Full Context ID match. DBGBVR&lt;-&gt;_EL1.ContextID is compared against CONTEXTIDR_EL1, and DBGBVR&lt;-&gt;_EL1.ContextID2 is compared against CONTEXTIDR_EL2.</td>
</tr>
<tr>
<td>0b1011</td>
<td>As 0b1100, with linking enabled.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Unlinked CONTEXTIDR_EL2 match. DBGBVR&lt;-&gt;_EL1.ContextID2 is a Context ID compared against CONTEXTIDR_EL2.</td>
</tr>
<tr>
<td>0b1101</td>
<td>As 0b1100, with linking enabled.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Unlinked Full Context ID match. DBGBVR&lt;-&gt;_EL1.ContextID is compared against CONTEXTIDR_EL1, and DBGBVR&lt;-&gt;_EL1.ContextID2 is compared against CONTEXTIDR_EL2.</td>
</tr>
<tr>
<td>0b1111</td>
<td>As 0b1110, with linking enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved. Constraints on breakpoint programming mean other values are reserved under some conditions.

For more information on the operation of the SSC, HMC, and PMC fields, and on the effect of programming this field to a reserved value, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug) and 'Reserved DBGBCR<->_EL1.BT values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**LBN, bits [19:16]**

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an UNKNOWN value.

This field is ignored when the value of DBGBCR<->_EL1.BE is 0.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.
**SSC, bits [15:14]**

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the \{HMC, SSC, PMC\} fields.

For more information on the operation of the SSC, HMC, and PMC fields, and the effect of programming the fields to a reserved set of values, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug) and 'Reserved DBGCR<n>_EL1.(SSC, HMC, PMC) values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**HMC, bit [13]**

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the \{HMC, SSC, PMC\} fields. For more information see the SSC, bits [15:14] description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**Bits [12:9]**

Reserved, RES0.

**BAS, bits [8:5]**

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state. In an AArch64 only implementation, this field is reserved, RES1.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Match instruction at</th>
<th>Constraint for debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0011</td>
<td>DBGBVR&lt;n&gt;_EL1</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1100</td>
<td>DBGBVR&lt;n&gt;_EL1 + 2</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1111</td>
<td>DBGBVR&lt;n&gt;_EL1</td>
<td>Use for A64 and A32 instructions</td>
</tr>
</tbody>
</table>

All other values are reserved. For more information, see 'Reserved DBGCR<n>_EL1.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

For more information on using the BAS field in address match breakpoints, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Context matching breakpoints, this field is RES1 and ignored.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.
Bits [4:3]
Reserved, RES0.

PMC, bits [2:1]
Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the DBGBCR<n>_EL1.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:
- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

E, bit [0]
Enable breakpoint DBGBVR<n>_EL1. Possible values are:

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Breakpoint disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Breakpoint enabled.</td>
</tr>
</tbody>
</table>

The following resets apply:
- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

Accessing the DBGBCR<n>_EL1

Accesses to this register use the following encodings:

MRS <Xt>, DBGBCR<n>_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.DBGBCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGBCR_EL1[UInt(CRm<3:0>)];
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBCR_EL1[UInt(CRm<3:0>)] = X[t];
  end if;
elsif PSTATE.EL == EL3 then
  if !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBCR_EL1[UInt(CRm<3:0>)] = X[t];
  end if;
else
  if PSTATE.EL == EL0 then
    return DBGBCR_EL1[UInt(CRm<3:0>)];
  elsif PSTATE.EL == EL1 then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBCR_EL1[UInt(CRm<3:0>)] = X[t];
  end if;
end if;

MSR DBGBCR<n>_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>
DBGBVR\textsubscript{n}\_EL1, Debug Breakpoint Value Registers, \textit{n} = 0 - 15

The DBGBVR\textsubscript{n}\_EL1 characteristics are:

**Purpose**

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint \textit{n} together with control register DBGBCR\textsubscript{n}\_EL1.

**Configuration**

AArch64 System register DBGBVR\textsubscript{n}\_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBVR\textsubscript{n}[31:0].

AArch64 System register DBGBVR\textsubscript{n}\_EL1 bits [63:32] are architecturally mapped to AArch32 System register DBGBXVR\textsubscript{n}[31:0].

AArch64 System register DBGBVR\textsubscript{n}\_EL1 bits [63:0] are architecturally mapped to External register DBGBVR\textsubscript{n}\_EL1[63:0].

If breakpoint \textit{n} is not implemented then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

How this register is interpreted depends on the value of DBGBCR\textsubscript{n}\_EL1.BT.

- When DBGBCR\textsubscript{n}\_EL1.BT is 0b000x, this register holds a virtual address.
- When DBGBCR\textsubscript{n}\_EL1.BT is 0b001x, 0b011x, or 0b110x, this register holds a Context ID.
- When DBGBCR\textsubscript{n}\_EL1.BT is 0b100x, this register holds a VMID.
- When DBGBCR\textsubscript{n}\_EL1.BT is 0b101x, this register holds a VMID and a Context ID.
- When DBGBCR\textsubscript{n}\_EL1.BT is 0b111x, this register holds two Context ID values.

For other values of DBGBCR\textsubscript{n}\_EL1.BT, this register is RES0.

**Field descriptions**

The DBGBVR\textsubscript{n}\_EL1 bit assignments are:

**When DBGBCR\textsubscript{n}\_EL1.BT == 0b000x:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
</table>

VA[48:2]

31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0

**RESS[14:4], bits [63:53]**

Reserved, Sign extended. Software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

It is IMPLEMENTATION DEFINED whether:

- Reads return the value of the most significant bit of the VA for every bit in this field.
- Reads return the last value written.
The PE ignores this field.

**VA[52:49], bits [52:49]**

When ARMv8.2-LVA is implemented:


The following resets apply:

- On a Cold reset, this field resets to an architecturally `UNKNOWN` value.
- On a Warm reset, the value of this field is unchanged.

Otherwise:


**VA[48:2], bits [48:2]**

Bits[48:2] of the address value for comparison.

When ARMv8.2-LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, VA[52:49] are RESS.

The following resets apply:

- On a Cold reset, this field resets to an architecturally `UNKNOWN` value.
- On a Warm reset, the value of this field is unchanged.

**Bits [1:0]**

Reserved, RES0.

**When DBGBCR<n>_EL1.BT == 0b001x:**

- `RES0` ContextID

**Bits [63:32]**

Reserved, RES0.

**ContextID, bits [31:0]**

Context ID value for comparison.

The value is compared against `CONTEXTIDR_EL2` when ARMv8.1-VHE is implemented, `HCR_EL2.E2H` is 1, and either:

- The PE is executing at EL2.
- `HCR_EL2.TGE` is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

Otherwise, the value is compared against `CONTEXTIDR_EL1`.

The following resets apply:

- On a Cold reset, this field resets to an architecturally `UNKNOWN` value.
- On a Warm reset, the value of this field is unchanged.
When DBGBCR<n>_EL1.BT == 0b011x:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ContextID |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**ContextID, bits [31:0]**

Context ID value for comparison against CONTEXTIDR_EL1.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

When DBGBCR<n>_EL1.BT == 0b100x and HaveEL(EL2):

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | VMID[15:8] | VMID[7:0] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:48]**

Reserved, RES0.

**VMID[15:8], bits [47:40]**

When ARMv8.1-VMID16 is implemented:

Extension to VMID[7:0]. See VMID[7:0] for more details.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**Otherwise:**

Reserved, RES0.

**VMID[7:0], bits [39:32]**

VMID value for comparison.

The VMID is 8 bits in the following cases.

- EL2 is using AArch32.
- ARMv8.1-VMID16 is not implemented.

When ARMv8.1-VMID16 is implemented and EL2 is using AArch64, it is IMPLEMENTATION DEFINED whether the VMID is 8 bits or 16 bits.

VMID[15:8] is RES0 if any of the following applies:

- The implementation has an 8-bit VMID.
• \texttt{VTCR\_EL2\_VS} has a value of 0.
• EL2 is using AArch32.

The following resets apply:

• On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.
• On a Warm reset, the value of this field is unchanged.

\textbf{Bits [31:0]}

Reserved, \texttt{RES0}.

\textbf{When DBGBCR\textless n\textgreater\_EL1\_BT == 0b101x and HaveEL(EL2):}

\begin{verbatim}
<table>
<thead>
<tr>
<th>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
</tr>
<tr>
<td>ContextID</td>
</tr>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>
\end{verbatim}

\textbf{Bits [63:48]}

Reserved, \texttt{RES0}.

\textbf{VMID[15:8], bits [47:40]}

\textbf{When ARMv8.1-VMID16 is implemented:}

Extension to VMID[7:0]. See VMID[7:0] for more details.

The following resets apply:

• On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.
• On a Warm reset, the value of this field is unchanged.

\textbf{Otherwise:}

Reserved, \texttt{RES0}.

\textbf{VMID[7:0], bits [39:32]}

VMID value for comparison.

The VMID is 8 bits in the following cases.

• EL2 is using AArch32.
• ARMv8.1-VMID16 is not implemented.

When ARMv8.1-VMID16 is implemented and EL2 is using AArch64, it is \texttt{IMPLEMENTATION DEFINED} whether the VMID is 8 bits or 16 bits.

VMID[15:8] is \texttt{RES0} if any of the following applies:

• The implementation has an 8-bit VMID.
• \texttt{VTCR\_EL2\_VS} has a value of 0.
• EL2 is using AArch32.

The following resets apply:

• On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.
• On a Warm reset, the value of this field is unchanged.
ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

When DBGBCR<n>_EL1.BT == 0b110x and HaveEL(EL2):

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ContextID2 | RES0 |

ContextID2, bits [63:32]

When ARMv8.1-VHE is implemented:

Context ID value for comparison against CONTEXTIDR_EL2.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

Otherwise:

Reserved, RES0.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b111x, HaveEL(EL2) and ARMv8.1-VHE is implemented:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ContextID2 | ContextID |

ContextID2, bits [63:32]

When ARMv8.1-VHE is implemented:

Context ID value for comparison against CONTEXTIDR_EL2.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

Otherwise:

Reserved, RES0.
Context ID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

Accessing the DBGVR<n>_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, DBGVR<n>_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.DDBGVRn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGVR_EL1[UInt(CRm<3:0>)];
else
  PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
      Halt(DebugHalt_SoftwareAccess);
    else
      return DBGVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
  if !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGVR_EL1[UInt(CRm<3:0>)];
```

**MSR DBGVR<n>_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.DBGVRn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elseif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBVR_EL1[UInt(CRm<3:0>)] = X[t];
  endelse
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elseif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBVR_EL1[UInt(CRm<3:0>)] = X[t];
  endelse
elsif PSTATE.EL == EL3 then
  if !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBVR_EL1[UInt(CRm<3:0>)] = X[t];
  endelse

27/09/2019 18:22; 6134483bd14dc8c12a99c984cb6e3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGCLAIMCLR_EL1, Debug CLAIM Tag Clear register

The DBGCLAIMCLR_EL1 characteristics are:

**Purpose**

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

---

**Note**

CLAIM tags are typically used for communication between the debugger and target software.

---

Used in conjunction with the DBGCLAIMSET_EL1 register.

**Configuration**

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMCLR[31:0].

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

An implementation must include eight CLAIM tag bits.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGCLAIMCLR_EL1 is a 64-bit register.

**Field descriptions**

The DBGCLAIMCLR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RAZ/SBZ</td>
</tr>
<tr>
<td>31</td>
<td>CLAIM</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**Bits [31:8]**

Reserved, RAZ/SBZ. Software can rely on these bits reading as zero, and must use a should-be-zero policy on writes. Implementations must ignore writes.

**CLAIM, bits [7:0]**

Read or clear CLAIM tag bits. Reading this field returns the current value of the CLAIM tag bits.
Writing a 1 to one of these bits clears the corresponding CLAIM tag bit to 0. This is an indirect write to the CLAIM tag bits. A single write operation can clear multiple CLAIM tag bits to 0.

Writing 0 to one of these bits has no effect.

On a Cold reset, this field resets to 0.

### Accessing the DBGCLAIMCLR_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, DBGCLAIMCLR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGCLAIM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccess Trap(EL2, 0x18);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return DBGCLAIMCLR_EL1;
  endif
else
  return DBGCLAIMCLR_EL1;
endif

**MSR DBGCLAIMCLR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.DBGCLAIM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    DBGCLAIMCLR_EL1 = X[t];
  endif
else
  return DBGCLAIMCLR_EL1;
endif

if PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    DBGCLAIMCLR_EL1 = X[t];
  endif
else
  return DBGCLAIMCLR_EL1;
endif

if PSTATE.EL == EL3 then
  DBGCLAIMCLR_EL1 = X[t];
else
  return DBGCLAIMCLR_EL1;
endif
DBGCLAIMSET_EL1, Debug CLAIM Tag Set register

The DBGCLAIMSET_EL1 characteristics are:

**Purpose**

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

---

**Note**

CLAIM tags are typically used for communication between the debugger and target software.

---

Used in conjunction with the [DBGCLAIMCLR_EL1](#) register.

**Configuration**

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGCLAIMSET][31:0].

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register [DBGCLAIMSET_EL1][31:0].

An implementation must include eight CLAIM tag bits.

**Attributes**

DBGCLAIMSET_EL1 is a 64-bit register.

**Field descriptions**

The DBGCLAIMSET_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 |
|      |

**Bits [63:32]**

Reserved, RES0.

**Bits [31:8]**

Reserved, RAZ/SBZ. Software can rely on these bits reading as zero, and must use a should-be-zero policy on writes. Implementations must ignore writes.

**CLAIM, bits [7:0]**

Set CLAIM tag bits.

This field is RAO.

Writing a 1 to one of these bits sets the corresponding CLAIM tag bit to 1. This is an indirect write to the CLAIM tag bits. A single write operation can set multiple CLAIM tag bits to 1.
Writing 0 to one of these bits has no effect.

**Accessing the DBGCLAIMSET_EL1**

Accesses to this register use the following encodings:

MRS `<Xt>`, DBGCLAIMSET_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGCLAIMSET_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGCLAIMSET_EL1;
        elsif PSTATE.EL == EL3 then
            return DBGCLAIMSET_EL1;
else
    if PSTATE.EL == EL0 then
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.DBGCLAIM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMSET_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGCLAIMSET_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    return DBGCLAIMSET_EL1;
```

MSR DBGCLAIMSET_EL1, `<Xt>`

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

...
DBGDTR_EL0, Debug Data Transfer Register, half-duplex

The DBGDTR_EL0 characteristics are:

**Purpose**

Transfers 64 bits of data between the PE and an external debugger. Can transfer both ways using only a single register.

**Configuration**

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register DBGDTRRX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0] when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTX_EL0[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRTXint[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register DBGDTRTX_EL0[31:0] when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0] when read.

**Attributes**

DBGDTR_EL0 is a 64-bit register.

**Field descriptions**

The DBGDTR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

<table>
<thead>
<tr>
<th>HighWord</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LowWord</td>
<td></td>
</tr>
</tbody>
</table>
HighWord, bits [63:32]

Writes to this register set DTRRX to the value in this field and do not change RXfull.

Reads of this register:
  • If RXfull is set to 1, return the last value written to DTRTX.
  • If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

LowWord, bits [31:0]

Writes to this register set DTRTX to the value in this field and set TXfull to 1.

Reads of this register:
  • If RXfull is set to 1, return the last value written to DTRRX.
  • If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

Accessing the DBGDTR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, DBGDTR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL1.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  end if
elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' ||
  MDCR_EL2.<TDE,TDA> != '00') then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDCC == '1' then
  AArch64.SystemAccessTrap(EL3, 0x18);
elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDA == '1' then
  AArch64.SystemAccessTrap(EL3, 0x18);
else
  return DBGDTR_EL0;
endif
elsif PSTATE.EL == EL1 then
  if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return DBGDTR_EL0;
  endif
elsif PSTATE.EL == EL2 then
  if !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return DBGDTR_EL0;
  endif
elsif PSTATE.EL == EL3 then
  return DBGDTR_EL0;
endif

MSR DBGDTR_EL0, <Xt>
if PSTATE_EL == EL0 then
    if !Halted() && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGDTR_EL0 = X[t];
    endif
else
    PSTATE_EL == EL1 then
        if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTR_EL0 = X[t];
        endif
    elsif PSTATE_EL == EL2 then
        if !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTR_EL0 = X[t];
        endif
    elsif PSTATE_EL == EL3 then
        DBGDTR_EL0 = X[t];
    endif
DBGDTRRX_EL0, Debug Data Transfer Register, Receive

The DBGDTRRX_EL0 characteristics are:

**Purpose**

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

**Configuration**

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGDTRRX_EL0 is a 64-bit register.

**Field descriptions**

The DBGDTRRX_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**Bits [31:0]**

Update DTRRX.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see The Debug Communication Channel and Instruction Transfer Register.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.
Accessing the DBGDTRRX_EL0

Accesses to this register use the following encodings:

MRS <Xt>, DBGDTRRX_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b11</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !Halted() && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        MDCR_EL1.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif !Halted() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
        MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif !Halted() && !ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return DBGDTRRX_EL0;
    elsif PSTATE.EL == EL1 then
        if !Halted() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTRRX_EL0;
    elsif PSTATE.EL == EL2 then
        if !Halted() && !ELUsingAArch32(EL3) && MDSCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTRRX_EL0;
    elsif PSTATE.EL == EL3 then
        return DBGDTRRX_EL0;
The DBGDTRTX_EL0 characteristics are:

**Purpose**

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communication Channel.

**Configuration**

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to External register DBGDTRTX_EL0[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGDTRTX_EL0 is a 64-bit register.

**Field descriptions**

The DBGDTRTX_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>31-0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**Bits [31:0]**

Return DTRTX.

Writes to this register:

- If TXfull is set to 1, set DTRRX and DTRTX to **UNKNOWN**.
- If TXfull is set to 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see The Debug Communication Channel and Instruction Transfer Register.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.
Accessing the DBGDTRTX_EL0

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if !Halted() && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    DBGDTRTX_EL0 = X[t];
  endif
else
  DBGDTRTX_EL0 = X[t];
endif
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGPRCR_EL1, Debug Power Control Register

The DBGPRCR_EL1 characteristics are:

**Purpose**

Controls behavior of the PE on powerdown request.

**Configuration**

AArch64 System register DBGPRCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGPRCR[31:0].

Bit [0] of this register is mapped to EDPRCR.CORENPDRQ, bit [0] of the external view of this register.

The other bits in these registers are not mapped to each other.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGPRCR_EL1 is a 64-bit register.

**Field descriptions**

The DBGPRCR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | CORENPDRQ |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:1]**

Reserved, RES0.

**CORENPDRQ, bit [0]**

*When ARMv8.3-DoPD is implemented:*

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the **IMPLEMENTATION DEFINED** nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>CORENPDRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the system responds to a powerdown request, it powers down Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.</td>
</tr>
</tbody>
</table>

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is **IMPLEMENTATION DEFINED** whether this bit is reset to its Cold reset value on exit from an **IMPLEMENTATION DEFINED** software-visible retention state. For more information about retention states see Core power domain power states.
Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The following resets apply:

- On a Cold reset, this field is set to 1 if the powerup request is implemented and the powerup request has been asserted, and is set to 0 otherwise.
- On a Warm reset, the value of this field is unchanged.

Otherwise:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>CORENPDRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the system responds to a powerdown request, it powers down Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.</td>
</tr>
</tbody>
</table>

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of EDPRCR.COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see Core power domain power states.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The following resets apply:

- On a Cold reset, this field resets to the value in EDPRCR.COREPURQ.
- On a Warm reset, the value of this field is unchanged.

Accessing the DBGPRCR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, DBGPRCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HDFGWTR_EL2 DBGPRCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return DBGPRCR_EL1;
elseif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return DBGPRCR_EL1;
elseif PSTATE.EL == EL3 then
  return DBGPRCR_EL1;

MSR DBGPRCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HDFGWTR_EL2 DBGPRCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    DBGPRCR_EL1 = X[t];
elseif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    DBGPRCR_EL1 = X[t];
elseif PSTATE.EL == EL3 then
  DBGPRCR_EL1 = X[t];
DBGVCR32_EL2, Debug Vector Catch Register

The DBGVCR32_EL2 characteristics are:

**Purpose**

Allows access to the AArch32 register DBGVCR from AArch64 state only. Its value has no effect on execution in AArch64 state.

**Configuration**

AArch64 System register DBGVCR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register DBGVCR[31:0].

If EL1 does not support AArch32, this register is **UNDEFINED**.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not **RES0**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGVCR32_EL2 is a 64-bit register.

**Field descriptions**

The DBGVCR32_EL2 bit assignments are:

**When HaveEL(EL3) and !ELUsingAArch32(EL3):**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NSF | NSI | NSF | NSD | NSF | NSP | NSF | NSU | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 | NSF | NS0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:32]**

Reserved, **RES0**.

**NSF, bit [31]**

FIQ vector catch enable in Non-secure state.

The exception vector offset is 0x1C.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**NSI, bit [30]**

IRQ vector catch enable in Non-secure state.

The exception vector offset is 0x18.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
Bit [29]
Reserved, RES0.

NSD, bit [28]
Data Abort vector catch enable in Non-secure state.
The exception vector offset is 0x10.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]
Prefetch Abort vector catch enable in Non-secure state.
The exception vector offset is 0x0C.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]
Supervisor Call (SVC) vector catch enable in Non-secure state.
The exception vector offset is 0x08.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]
Undefined Instruction vector catch enable in Non-secure state.
The exception vector offset is 0x04.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:8]
Reserved, RES0.

SF, bit [7]
FIQ vector catch enable in Secure state.
The exception vector offset is 0x1C.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]
IRQ vector catch enable in Secure state.
The exception vector offset is 0x18.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]
Reserved, RES0.
SD, bit [4]

Data Abort vector catch enable in Secure state.
The exception vector offset is 0x10.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.
The exception vector offset is 0x0C.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.
The exception vector offset is 0x08.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.
The exception vector offset is 0x04.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When !HaveEL(EL3):

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>F</td>
<td>I</td>
<td>RES0</td>
<td>D</td>
<td>P</td>
<td>S</td>
<td>U</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:8]

Reserved, RES0.

F, bit [7]

FIQ vector catch enable.
The exception vector offset is 0x1C.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [6]

IRQ vector catch enable.
The exception vector offset is 0x18.
On a Warm reset, this field resets to an architecturally UNKNOWN value.
Bit [5]

Reserved, RES0.

D, bit [4]

Data Abort vector catch enable.

The exception vector offset is 0x10.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

P, bit [3]

Prefetch Abort vector catch enable.

The exception vector offset 0x0C.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [2]

Supervisor Call (SVC) vector catch enable.

The exception vector offset 0x08.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [1]

Undefined Instruction vector catch enable.

The exception vector offset 0x04.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

**Accessing the DBGVCR32_EL2**

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ DBGVCR32_EL2}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return DBGVCR32_EL2;
elsif PSTATE.EL == EL3 then
  return DBGVCR32_EL2;

MSR DBGVCR32_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    DBGVCR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  DBGVCR32_EL2 = X[t];
DBGWCR<n>_EL1, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR<n>_EL1 characteristics are:

**Purpose**

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGVR<n>_EL1.

**Configuration**

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWVR<n>[31:0].

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to External register DBGWCR<n>_EL1[31:0].

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGWCR<n>_EL1 is a 64-bit register.

**Field descriptions**

The DBGWCR<n>_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:29]**

Reserved, RES0.

**MASK, bits [28:24]**

Address mask. Only objects up to 2GB can be watched using a single mask.

<table>
<thead>
<tr>
<th>MASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>No mask.</td>
</tr>
<tr>
<td>0b00001</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

If programmed with a reserved value, a watchpoint must behave as if either:

- MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCRn_EL1.
- The watchpoint is disabled.

Software must not rely on this property because the behavior of reserved values might change in a future revision of the architecture.
Other values mask the corresponding number of address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**Bits [23:21]**

Reserved, **RES0**.

**WT, bit [20]**

Watchpoint type. Possible values are:

<table>
<thead>
<tr>
<th>WT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unlinked data address match.</td>
</tr>
<tr>
<td>0b1</td>
<td>Linked data address match.</td>
</tr>
</tbody>
</table>

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**LBN, bits [19:16]**

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**SSC, bits [15:14]**

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug), and 'Reserved DBGBCR<n>_EL1. {SSC, HMC, PMC} values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**HMC, bit [13]**

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
• On a Warm reset, the value of this field is unchanged.

**BAS, bits [12:5]**

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by \texttt{DBGWVR<n>_EL1} is being watched.

<table>
<thead>
<tr>
<th>BAS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxxxxx1</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1}</td>
</tr>
<tr>
<td>xxxxxx1x</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 1</td>
</tr>
<tr>
<td>xxxxx1xx</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 2</td>
</tr>
<tr>
<td>xxx1xxx</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 3</td>
</tr>
</tbody>
</table>

In cases where \texttt{DBGWVR<n>_EL1} addresses a double-word:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Description, if \texttt{DBGWVR&lt;n&gt;_EL1}[2] == 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx1xxxx</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 4</td>
</tr>
<tr>
<td>xx1xxxxx</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 5</td>
</tr>
<tr>
<td>x1xxxxxx</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 6</td>
</tr>
<tr>
<td>1xxxxxxx</td>
<td>Match byte at \texttt{DBGWVR&lt;n&gt;_EL1} + 7</td>
</tr>
</tbody>
</table>

If \texttt{DBGWVR<n>_EL1}[2] == 1, only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting \texttt{DBGWVR<n>_EL1}[2] == 1.

The valid values for BAS are non-zero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>_EL1.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**LSC, bits [4:3]**

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

<table>
<thead>
<tr>
<th>LSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>Match instructions that load from a watchpointed address.</td>
</tr>
<tr>
<td>0b10</td>
<td>Match instructions that store to a watchpointed address.</td>
</tr>
<tr>
<td>0b11</td>
<td>Match instructions that load from or store to a watchpointed address.</td>
</tr>
</tbody>
</table>

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**PAC, bits [2:1]**

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.
E, bit [0]

Enable watchpoint n. Possible values are:

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint enabled.</td>
</tr>
</tbody>
</table>

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

### Accessing the DBGWCR<n>_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, DBGWCR<n>_EL1**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b111</td>
</tr>
</tbody>
</table>
```

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2(DBGWCRn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWCR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWCR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
  if !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWCR_EL1[UInt(CRm<3:0>)];
```

**MSR DBGWCR<n>_EL1, <Xt>**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b111</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.DBGWCRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR_EL1[UInt(CRm<3:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGWCR_EL1[UInt(CRm<3:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
        if !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGWCR_EL1[UInt(CRm<3:0>)] = X[t];

DBGWVR<n>_EL1, Debug Watchpoint Value Registers, n = 0 - 15

The DBGWVR<n>_EL1 characteristics are:

**Purpose**

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register DBGWCR<n>_EL1.

**Configuration**

AArch64 System register DBGWVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWVR<n>[31:0].

AArch64 System register DBGWVR<n>_EL1 bits [63:0] are architecturally mapped to External register DBGWVR<n>_EL1[63:0].

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGWVR<n>_EL1 is a 64-bit register.

**Field descriptions**

The DBGWVR<n>_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**RESS[14:4], bits [63:53]**

Reserved, Sign extended. Hardware and software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:

- The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.
- The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.

**VA[52:49], bits [52:49]**

When ARMv8.2-LVA is implemented:


The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.
Otherwise:


VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When ARMv8.2-LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, VA[52:49] are RESS.


The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

Bits [1:0]

Reserved, RES0.

Accessing the DBGWVR<n>_EL1

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGWVRn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
  if !ELUsingAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWVR_EL1[UInt(CRm<3:0>)];
else
  return DBGWVR_EL1[UInt(CRm<3:0>)];
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
       HDFGWTR_EL2.DBGWVRn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !ELUsingAAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR_EL1[UInt(CRm<3:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !ELUsingAAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGWVR_EL1[UInt(CRm<3:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
        if !ELUsingAAArch32(EL1) && OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGWVR_EL1[UInt(CRm<3:0>)] = X[t];
endif

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DC CGDSW, Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way

The DC CGDSW characteristics are:

**Purpose**

Clean data and Allocation Tags in data cache by set/way.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC CGDSW are **UNDEFINED**.

**Attributes**

DC CGDSW is a 64-bit System instruction.

**Field descriptions**

The DC CGDSW input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    | RES0 |    |    |    |    |    |    |    | SetWay |    |    |    |    |    |    |    |    |    |    |    | Level |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[1:4] are RES0.

\[ A = \log_2(\text{ASSOCIATIVITY}), \quad L = \log_2(\text{LINELEN}), \quad B = (L + S), \quad S = \log_2(\text{NSETS}). \]

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.
Executing the DC CGDSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

DC CGDSW, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCCSW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CGDSW(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  DC_CGDSW(X[t]);
elsif PSTATE.EL == EL3 then
  DC_CGDSW(X[t]);
endif
DC CGDVAC, Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC

The DC CGDVAC characteristics are:

**Purpose**

Clean data and Allocation Tags in data cache by address to Point of Coherency.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CGDVAC are UNDEFINED.

**Attributes**

DC CGDVAC is a 64-bit System instruction.

**Field descriptions**

The DC CGDVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Virtual address to use**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CGDVAC instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

\[
\text{DC CGDVAC, } <\text{Xt}> \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CGDVAC(X[t]);
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      DC_CGDVAC(X[t]);
  elsif PSTATE.EL == EL2 then
    DC_CGDVAC(X[t]);
  elsif PSTATE.EL == EL3 then
    DC_CGDVAC(X[t]);
DC CGDVADP, Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoDP

The DC CGDVADP characteristics are:

**Purpose**

Clean Allocation Tags in data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a DC CGVAP.

**Configuration**

This instruction is present only when ARMv8.2-DCCVADP is implemented and ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CGDVADP are UNDEFINED.

**Attributes**

DC CGDVADP is a 64-bit System instruction.

**Field descriptions**

The DC CGDVADP input value bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| Virtual address to use | Virtual address to use |

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CGDVADP instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

\[
\text{DC CGDVADP, \ <Xt>}
\]

<table>
<thead>
<tr>
<th>(\text{op0})</th>
<th>(\text{op1})</th>
<th>(\text{CRn})</th>
<th>(\text{CRm})</th>
<th>(\text{op2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1101</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CGDVADP(X[t]);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_CGDVADP(X[t]);
        elsif PSTATE.EL == EL2 then
            DC_CGDVADP(X[t]);
        elsif PSTATE.EL == EL3 then
            DC_CGDVADP(X[t]);
    end

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DC CGDVAP, Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoP

The DC CGDVAP characteristics are:

**Purpose**

Clean data and Allocation Tags in data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a DC CGDVAC.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CGDVAP are UNDEFINED.

**Attributes**

DC CGDVAP is a 64-bit System instruction.

**Field descriptions**

The DC CGDVAP input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>31</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CGDVAP instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

DC CGDVAP, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CGDVAP(X[t]);
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      DC_CGDVAP(X[t]);
  elsif PSTATE.EL == EL2 then
    DC_CGDVAP(X[t]);
else
  if PSTATE.EL == EL3 then
    DC_CGDVAP(X[t]);
else
  DC_CGDVAP(X[t]);

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The DC CGSW characteristics are:

**Purpose**

Clean Allocation Tags in data cache by set/way.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC CGSW are **UNDEFINED**.

**Attributes**

DC CGSW is a 64-bit System instruction.

**Field descriptions**

The DC CGSW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31-4</td>
<td>SetWay, bits [31:4]</td>
</tr>
<tr>
<td>3-0</td>
<td>Level, bits [3:1]</td>
</tr>
<tr>
<td>0</td>
<td>Bit [0] Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = \( \log_2(\text{ASSOCIATIVITY})\), L = \( \log_2(\text{LINELEN})\), B = (L + S), S = \( \log_2(\text{NSETS})\).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.
Executing the DC CGSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

DC CGSW, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CGSW(X[t]);
    elsif PSTATE.EL == EL2 then
        DC_CGSW(X[t]);
    elsif PSTATE.EL == EL3 then
        DC_CGSW(X[t]);
    else
        DC_CGSW(X[t]);

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DC CGVAC, Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC

The DC CGVAC characteristics are:

**Purpose**

Clean Allocation Tags in data cache by address to Point of Coherency.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CGVAC are UNDEFINED.

**Attributes**

DC CGVAC is a 64-bit System instruction.

**Field descriptions**

The DC CGVAC input value bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     | Virtual address to use |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CGVAC instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see ‘The data cache maintenance instruction (DC)’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

$$ DC \text{ CGVAC, } <Xt> $$

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CGVAC(X[t]);
  endif
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CGVAC(X[t]);
  endif
elseif PSTATE.EL == EL2 then
  DC_CGVAC(X[t]);
elsif PSTATE.EL == EL3 then
  DC_CGVAC(X[t]);
endif
DC CGVADP, Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoDP

The DC CGVADP characteristics are:

**Purpose**

Clean data and Allocation Tags in data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a DC CGDVAP.

**Configuration**

This instruction is present only when ARMv8.2-DCCVADP is implemented and ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CGVADP are UNDEFINED.

**Attributes**

DC CGVADP is a 64-bit System instruction.

**Field descriptions**

The DC CGVADP input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Bit assignments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>Virtual address to use</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CGVADP instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

DC CGVADP, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1101</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CGVADP(X[t]);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_CGVADP(X[t]);
    elsif PSTATE.EL == EL2 then
        DC_CGVADP(X[t]);
    elsif PSTATE.EL == EL3 then
        DC_CGVADP(X[t]);
    endif
else
    DC_CGVADP(X[t]);
endif
**DC CGVAP, Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP**

The DC CGVAP characteristics are:

**Purpose**

Clean Allocation Tags in data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a [DC CGVAC](#).

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CGVAP are **UNDEFINED**.

**Attributes**

DC CGVAP is a 64-bit System instruction.

**Field descriptions**

The DC CGVAP input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Virtual address to use**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CGVAP instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

**DC CGVAP, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_CGVAP(X[t]);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CGVAP(X[t]);
    endif
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CGVAP(X[t]);
    endif
elseif PSTATE.EL == EL2 then
    DC_CGVAP(X[t]);
elseif PSTATE.EL == EL3 then
    DC_CGVAP(X[t]);
**DC CIGDSW, Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by Set/Way**

The DC CIGDSW characteristics are:

**Purpose**

Clean and Invalidate data and Allocation Tags in data cache by set/way.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC CIGDSW are UNDEFINED.

**Attributes**

DC CIGDSW is a 64-bit System instruction.

**Field descriptions**

The DC CIGDSW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

\[
A = \log_2(ASSOCIATIVITY), \quad L = \log_2(LINELEN), \quad B = (L + S), \quad S = \log_2(NSETS).
\]

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.
Executing the DC CIGDSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

DC CIGDSW, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b111</td>
<td>0b110</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CIGDSW(X[t]);
    endif
elsif PSTATE.EL == EL2 then
    DC_CIGDSW(X[t]);
elsif PSTATE.EL == EL3 then
    DC_CIGDSW(X[t]);
```

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The DC CIGDVAC characteristics are:

**Purpose**

Clean and Invalidate data and Allocation Tags in data cache by address to Point of Coherency.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CIGDVAC are UNDEFINED.

**Attributes**

DC CIGDVAC is a 64-bit System instruction.

**Field descriptions**

The DC CIGDVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>Virtual address to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Executing the DC CIGDVAC instruction**

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

**DC CIGDVAC, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  endif
else
  DC_CIGDVAC(X[t]);
endif

if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CIGDVAC(X[t]);
  endif
endif

if PSTATE.EL == EL2 then
  DC_CIGDVAC(X[t]);
endif

if PSTATE.EL == EL3 then
  DC_CIGDVAC(X[t]);
endif
The DC CIGSW characteristics are:

### Purpose

Clean and Invalidate Allocation Tags in data cache by set/way.

### Configuration

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC CIGSW are UNDEFINED.

### Attributes

DC CIGSW is a 64-bit System instruction.

### Field descriptions

The DC CIGSW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

#### Bits [63:32]

Reserved, RES0.

#### SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

\[ A = \log_2(\text{ASSOCIATIVITY}), \quad L = \log_2(\text{LINELEN}), \quad B = (L + S), \quad S = \log_2(\text{NSETS}). \]

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

#### Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

#### Bit [0]

Reserved, RES0.
Executing the DC CIGSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

**DC CIGSW, \(<Xt>\)**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCCISW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CIGSW(X[t]);
  end
elsif PSTATE.EL == EL2 then
  DC_CIGSW(X[t]);
elsif PSTATE.EL == EL3 then
  DC_CIGSW(X[t]);
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DC CIGVAC, Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by VA to PoC

The DC CIGVAC characteristics are:

**Purpose**

Clean and Invalidate Allocation Tags in data cache by address to Point of Coherency.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC CIGVAC are UNDEFINED.

**Attributes**

DC CIGVAC is a 64-bit System instruction.

**Field descriptions**

The DC CIGVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>31</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CIGVAC instruction**

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

<table>
<thead>
<tr>
<th>DC CIGVAC, &lt;Xt&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
</tr>
<tr>
<td>0b01</td>
</tr>
</tbody>
</table>

Virtual address to use. No alignment restrictions apply to this VA.
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CIGVAC(X[t]);
    endif
else PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CIGVAC(X[t]);
    endif
else PSTATE.EL == EL2 then
    DC_CIGVAC(X[t]);
else PSTATE.EL == EL3 then
    DC_CIGVAC(X[t]);
end if
DC CISW, Data or unified Cache line Clean and Invalidate by Set/Way

The DC CISW characteristics are:

**Purpose**

Clean and Invalidate data cache by set/way.

When ARMv8.5-MemTag is implemented, this instruction might clean and invalidate Allocation Tags from caches.

**Configuration**

AArch64 System instruction DC CISW performs the same function as AArch32 System instruction DCCISW.

**Attributes**

DC CISW is a 64-bit System instruction.

**Field descriptions**

The DC CISW input value bit assignments are:

| Bits [63:32] | Reserved, RES0. |
| SetWay, bits [31:4] | Contains two fields: |
| | • Way, bits[31:32-A], the number of the way to operate on. |
| | • Set, bits[B-1:L], the number of the set to operate on. |
| | Bits[L-1:4] are RES0. |
| | A = Log2(ASSOCIATIVITY), L = Log2(LINELEN), B = (L + S), S = Log2(NSETS). |
| | ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer. |
| Level, bits [3:1] | Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache. |
| Bit [0] | Reserved, RES0. |
### Executing the DC CISW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is **CONSTRAINED UNPREDICTABLE** and one of the following occurs:

- The instruction is **UNDEFINED**.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

**DC CISW, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.DC_CISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CISW(X[t]);
    end
elsif PSTATE.EL == EL2 then
    DC_CISW(X[t]);
elsif PSTATE.EL == EL3 then
    DC_CISW(X[t]);
else
    DC_CISW(X[t]);
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DC CIVAC, Data or unified Cache line Clean and Invalidate by VA to PoC

The DC CIVAC characteristics are:

### Purpose

Clean and Invalidate data cache by address to Point of Coherency.

When ARMv8.5-MemTag is implemented, this instruction might clean and invalidate Allocation Tags from caches.

### Configuration

AArch64 System instruction DC CIVAC performs the same function as AArch32 System instruction DCCIMVAC.

### Attributes

DC CIVAC is a 64-bit System instruction.

### Field descriptions

The DC CIVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>31</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

### Executing the DC CIVAC instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see ‘The data cache maintenance instruction (DC)’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in ‘Permission fault’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

**DC CIVAC, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CIVAC(X[t]);
    end if
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_CIVAC(X[t]);
        end if
    elsif PSTATE.EL == EL2 then
        DC_CIVAC(X[t]);
    elsif PSTATE.EL == EL3 then
        DC_CIVAC(X[t]);
    end if

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DC CSW, Data or unified Cache line Clean by Set/Way

The DC CSW characteristics are:

**Purpose**

Clean data cache by set/way.

When ARMv8.5-MemTag is implemented, this instruction might clean Allocation Tags from caches.

**Configuration**

AArch64 System instruction DC CSW performs the same function as AArch32 System instruction DCSSW.

**Attributes**

DC CSW is a 64-bit System instruction.

**Field descriptions**

The DC CSW input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Set/Way</td>
<td>Level</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = \( \log_2(\text{ASSOCIATIVITY}) \), L = \( \log_2(\text{LINELEN}) \), B = (L + S), S = \( \log_2(\text{NSETS}) \).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.
Executing the DC CSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

DC CSW, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CSW(X[t]);
    endif
elsif PSTATE.EL == EL2 then
    DC_CSW(X[t]);
elsif PSTATE.EL == EL3 then
    DC_CSW(X[t]);
else
    DC_CSW(X[t]);

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CSW(X[t]);
    endif
elsif PSTATE.EL == EL2 then
    DC_CSW(X[t]);
elsif PSTATE.EL == EL3 then
    DC_CSW(X[t]);
else
    DC_CSW(X[t]);
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**DC CVAC, Data or unified Cache line Clean by VA to PoC**

The DC CVAC characteristics are:

**Purpose**

Clean data cache by address to Point of Coherency.

When ARMv8.5-MemTag is implemented, this instruction might clean Allocation Tags from caches.

**Configuration**

AArch64 System instruction DC CVAC performs the same function as AArch32 System instruction **DCCMVAC**.

**Attributes**

DC CVAC is a 64-bit System instruction.

**Field descriptions**

The DC CVAC input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CVAC instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

**DC CVAC, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CVAC(X[t]);
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCCVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    DC_CVAC(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  DC_CVAC(X[t]);
elsif PSTATE.EL == EL3 then
  DC_CVAC(X[t]);
end if
DC CVADP, Data or unified Cache line Clean by VA to PoDP

The DC CVADP characteristics are:

**Purpose**

Clean data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a [DC CVAP](#).

When ARMv8.5-MemTag is implemented, this instruction might clean Allocation Tags from caches.

**Configuration**

This instruction is present only when ARMv8.2-DCCVADP is implemented. Otherwise, direct accesses to DC CVADP are UNDEFINED.

**Attributes**

DC CVADP is a 64-bit System instruction.

**Field descriptions**

The DC CVADP input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CVADP instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

DC CVADP, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && !HaveEL(EL3) || SCR_EL3.FGTEn == '1' && HFGITR_EL2.DCCVADP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CVADP(X[t]);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_CVADP(X[t]);
    elsif PSTATE.EL == EL2 then
        DC_CVADP(X[t]);
    elsif PSTATE.EL == EL3 then
        DC_CVADP(X[t]);

The DC CVAP characteristics are:

**Purpose**

Clean data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a DC CVAC.

When ARMv8.5-MemTag is implemented, this instruction might clean Allocation Tags from caches.

**Configuration**

This instruction is present only when ARMv8.2-DCPoP is implemented. Otherwise, direct accesses to DC CVAP are UNDEFINED.

**Attributes**

DC CVAP is a 64-bit System instruction.

**Field descriptions**

The DC CVAP input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Virtual address to use |
| Virtual address to use |

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC CVAP instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

**DC CVAP, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL1.UCI == '0' then
        DC_CVAP(X[t]);
    else
        if !EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CVAP(X[t]);
    end if
elsif PSTATE.EL == EL1 then
    if !EL2Enabled() && HCR_EL2.TPCP == '1' then
        DC_CVAP(X[t]);
    else
        if !EL2Enabled() && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end if
    end if
elsif PSTATE.EL == EL2 then
    DC_CVAP(X[t]);
elsif PSTATE.EL == EL3 then
    DC_CVAP(X[t]);
DC CVAU, Data or unified Cache line Clean by VA to PoU

The DC CVAU characteristics are:

**Purpose**

Clean data cache by address to Point of Unification.

**Configuration**

AArch64 System instruction DC CVAU performs the same function as AArch32 System instruction [DCCMVAU](#).

**Attributes**

DC CVAU is a 64-bit System instruction.

**Field descriptions**

The DC CVAU input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Executing the DC CVAU instruction**

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

<table>
<thead>
<tr>
<th>DC CVAU, &lt;Xt&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
</tr>
<tr>
<td>0b01</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CVAU(X[t]);
    end if;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_CVAU(X[t]);
    end if;
elsif PSTATE.EL == EL2 then
    DC_CVAU(X[t]);
elsif PSTATE.EL == EL3 then
    DC_CVAU(X[t]);
end if;

DC CVAU, Data or unified Cache line Clean by VA to PoU
The DC GVA characteristics are:

**Purpose**

Write a value to the Allocation Tags of a naturally aligned block of N bytes, where the size of N is identified in DCZID_EL0. The Allocation Tag used is determined by the input address.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC GVA are UNDEFINED.

**Attributes**

DC GVA is a 64-bit System instruction.

**Field descriptions**

The DC GVA input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtual address to use</td>
<td>Virtual address to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

**Executing the DC GVA instruction**

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous data abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is not set.

If the memory region being zeroed is any type of Device memory, this instruction can give an alignment fault which is prioritized in the same way as other alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte within the block being accessed, and so it:

- Generates a Permission Fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings:

**DC GVA, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if \texttt{PSTATE.EL == EL0} then
  if \texttt{!ELUsingAArch32(EL1) \&\& !EL2Enabled() \&\& HCR\_EL2.<E2H,TGE> == '11' \&\& SCTLR\_EL1.DZE == '0'} then
    if \texttt{EL2Enabled()} \&\& !ELUsingAArch32(EL2) \&\& HCR\_EL2.TGE == '1' then
      \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
    else
      \texttt{AArch64.SystemAccessTrap(EL1, 0x18)};
    end if
  end if
elsif \texttt{EL2Enabled()} \&\& !ELUsingAArch32(EL2) \&\& HCR\_EL2.<E2H,TGE> != '11' \&\& HCR\_EL2.TDZ == '1' then
  \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
elsif \texttt{EL2Enabled()} \&\& !ELUsingAArch32(EL2) \&\& HCR\_EL2.<E2H,TGE> == '11' \&\& HCR\_EL2.TDZ == '0' then
  \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
else
  \texttt{DC\_GVA(X[t])};
elsif \texttt{PSTATE.EL == EL1} then
  if \texttt{EL2Enabled()} \&\& !ELUsingAArch32(EL2) \&\& HCR\_EL2.TDZ == '1' then
    \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
  else
    \texttt{DC\_GVA(X[t])};
  end if
elsif \texttt{PSTATE.EL == EL2} then
  \texttt{DC\_GVA(X[t])};
elsif \texttt{PSTATE.EL == EL3} then
  \texttt{DC\_GVA(X[t])};
DC GZVA, Data Cache set Allocation Tags and Zero by VA

The DC GZVA characteristics are:

**Purpose**

Zero data and write a value to the Allocation Tags of a naturally aligned block of N bytes, where the size of N is identified in DCZID_EL0. The Allocation Tag used is determined by the input address.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to DC GZVA are UNDEFINED.

**Attributes**

DC GZVA is a 64-bit System instruction.

**Field descriptions**

The DC GZVA input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Virtual address to use
| Virtual address to use
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**Bits [63:0]**

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

**Executing the DC GZVA instruction**

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous data abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is not set.

If the memory region being zeroed is any type of Device memory, this instruction can give an alignment fault which is prioritized in the same way as other alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte within the block being accessed, and so it:

- Generates a Permission Fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings:

**DC GZVA, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TDZ == '1'
        then
            AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.DZE == '0'
        then
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_GZVA(X[t]);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.DZE == '0'
            then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_GZVA(X[t]);
    elsif PSTATE.EL == EL2 then
        DC_GZVA(X[t]);
    elsif PSTATE.EL == EL3 then
        DC_GZVA(X[t]);
DC IGDSW, Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way

The DC IGDSW characteristics are:

**Purpose**

Invalidate data and Allocation Tags in data cache by set/way.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC IGDSW are undefined.

**Attributes**

DC IGDSW is a 64-bit System instruction.

**Field descriptions**

The DC IGDSW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

\[
A = \log_2(\text{ASSOCIATIVITY}), \quad L = \log_2(\text{LINELEN}), \quad B = (L + S), \quad S = \log_2(\text{NSETS}).
\]

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.
Executing the DC IGDSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

DC IGDSW, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TSW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCISW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.SWIO == '1' then
    DC_CIGDSW(X[t]);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DC_CIGDSW(X[t]);
  else
    DC_IGDSW(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  DC_IGDSW(X[t]);
elsif PSTATE.EL == EL3 then
  DC_IGDSW(X[t]);
endif
The DC IGDVAC characteristics are:

**Purpose**

Invalidate data and Allocation Tags in data cache by address to Point of Coherency.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC IGDVAC are UNDEFINED.

**Attributes**

DC IGDVAC is a 64-bit System instruction.

**Field descriptions**

The DC IGDVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Virtual address to use

Virtual address to use

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC IGDVAC instruction**

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

**DC IGDVAC, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
        DC_CIGDVAC(X[t]);
    else
        DC_IGDVAC(X[t]);
    end if
else
    if PSTATE.EL == EL2 then
        DC_IGDVAC(X[t]);
    elsif PSTATE.EL == EL3 then
        DC_IGDVAC(X[t]);
    end if
end if
DC IGSW, Data, Allocation Tag or unified Cache line
 Invalidate of Allocation Tags by Set/Way

The DC IGSW characteristics are:

**Purpose**

Invalidate Allocation Tags in data cache by set/way.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC IGSW are **UNDEFINED**.

**Attributes**

DC IGSW is a 64-bit System instruction.

**Field descriptions**

The DC IGSW input value bit assignments are:

<p>|   | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |</p>
<table>
<thead>
<tr>
<th><strong>RES0</strong></th>
<th><strong>SetWay</strong></th>
<th><strong>Level</strong></th>
<th><strong>RES0</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, **RES0**.

**SetWay, bits [31:4]**

Contains two fields:
- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are **RES0**.

A = Log₂(ASSOCIATIVITY), L = Log₂(LINELEN), B = (L + S), S = Log₂(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, **RES0**.

---

Page 388
Executing the DC IGSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

DC IGSW, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCISW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.SWIO == '1' then
    DC_CIGSW(X[t]);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DC_CIGSW(X[t]);
  else
    DC_IGSW(X[t]);
  elsif PSTATE.EL == EL2 then
    DC_IGSW(X[t]);
  elsif PSTATE.EL == EL3 then
    DC_IGSW(X[t]);
  else
    DC_IGSW(X[t]);
```
The DC IGVAC characteristics are:

**Purpose**

Invalidate Allocation Tags in data cache by address to Point of Coherency.

**Configuration**

This instruction is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to DC IGVAC are **UNDEFINED**.

**Attributes**

DC IGVAC is a 64-bit System instruction.

**Field descriptions**

The DC IGVAC input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Virtual address to use | Virtual address to use |

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC IGVAC instruction**

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

\[ \text{DC IGVAC, } <Xt> \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCIVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DC_CIGVAC(X[t]);
  else
    DC_IGVAC(X[t]);
  end
elsif PSTATE.EL == EL2 then
  DC_IGVAC(X[t]);
elsif PSTATE.EL == EL3 then
  DC_IGVAC(X[t]);
DC ISW, Data or unified Cache line Invalidate by Set/Way

The DC ISW characteristics are:

**Purpose**

Invalidate data cache by set/way.

When ARMv8.5-MemTag is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

**Configuration**

AArch64 System instruction DC ISW performs the same function as AArch32 System instruction `DCISW`.

**Attributes**

DC ISW is a 64-bit System instruction.

**Field descriptions**

The DC ISW input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**Bits [63:32]**

Reserved, RES0.

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = \(\log_2(\text{ASSOCIATIVITY})\), L = \(\log_2(\text{LINELEN})\), B = (L + S), S = \(\log_2(\text{NSETS})\).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.
Executing the DC ISW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

**DC ISW, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCISW == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.SWIO == '1' then
    DC_CISW(X[t]);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DC_CISW(X[t]);
  else
    if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TSW == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.SWIO == '1' then
      DC_CISW(X[t]);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
      DC_CISW(X[t]);
    else
      DC_ISW(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  DC_ISW(X[t]);
elsif PSTATE.EL == EL3 then
  DC_ISW(X[t]);
else
  DC_ISW(X[t]);
endif
DC IVAC, Data or unified Cache line Invalidate by VA to PoC

The DC IVAC characteristics are:

**Purpose**

Invalidate data cache by address to Point of Coherency.

When ARMv8.5-MemTag is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

**Configuration**

AArch64 System instruction DC IVAC performs the same function as AArch32 System instruction DCIMVAC.

**Attributes**

DC IVAC is a 64-bit System instruction.

**Field descriptions**

The DC IVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>62</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>31</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>30</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>29</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>28</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>27</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>26</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>25</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>24</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>23</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>22</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>21</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>20</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>19</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>18</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>17</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>16</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>15</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>14</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>13</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>12</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>11</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>10</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>9</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>8</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>7</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>6</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>5</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>4</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>3</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>2</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>1</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>0</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DC IVAC instruction**

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission Fault, subject to the constraints described in 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.DCIVAC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DC_CIVAC(X[t]);
  else
    DC_IVAC(X[t]);
  end if
elsif PSTATE.EL == EL2 then
  DC_IVAC(X[t]);
elsif PSTATE.EL == EL3 then
  DC_IVAC(X[t]);
The DC ZVA characteristics are:

**Purpose**

Zero data cache by address. Zeroes a naturally aligned block of N bytes, where the size of N is identified in DCZID_EL0.

**Configuration**

**Attributes**

DC ZVA is a 64-bit System instruction.

**Field descriptions**

The DC ZVA input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Virtual address to use**

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

**Executing the DC ZVA instruction**

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous data abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 0.

If the memory region being zeroed is any type of Device memory, this instruction can give an Alignment fault which is prioritized in the same way as other Alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte within the block being accessed, and so it:

- Generates a Permission Fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings:

**DC ZVA, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TDZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.DZE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        DC_ZVA(X[t]);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            DC_ZVA(X[t]);
        elsif PSTATE.EL == EL2 then
            DC_ZVA(X[t]);
        elsif PSTATE.EL == EL3 then
            DC_ZVA(X[t]);
    else
        AArch64.SystemAccessTrap(EL2, 0x18);
    endif
DCZID_EL0, Data Cache Zero ID register

The DCZID_EL0 characteristics are:

**Purpose**

Indicates the block size that is written with byte values of 0 by the DC ZVA (Data Cache Zero by Address) System instruction.

If ARMv8.5-MemTag is implemented, this register also indicates the granularity at which the DC GVA and DC GZVA instructions write.

**Configuration**

**Attributes**

DCZID_EL0 is a 64-bit register.

**Field descriptions**

The DCZID_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:5</td>
<td>Reserved, RES0.</td>
<td></td>
</tr>
<tr>
<td>31:0</td>
<td>DZP</td>
<td>Data Zero Prohibited. This field indicates whether use of DC ZVA instructions is permitted or prohibited.</td>
</tr>
<tr>
<td></td>
<td>BS</td>
<td>Log2 of the block size in words. The maximum size supported is 2KB (value == 9).</td>
</tr>
</tbody>
</table>

**Bits [63:5]**

Reserved, RES0.

**DZP, bit [4]**

Data Zero Prohibited. This field indicates whether use of DC ZVA instructions is permitted or prohibited.

If ARMv8.5-MemTag is implemented, this field also indicates whether use of the DC GVA and DC GZVA instructions are permitted or prohibited.

<table>
<thead>
<tr>
<th>DZP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instructions are permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instructions are prohibited.</td>
</tr>
</tbody>
</table>

The value read from this field is governed by the access state and the values of the HCR_EL2.TDZ and SCTLR_EL1.DZE bits.

**BS, bits [3:0]**

Log2 of the block size in words. The maximum size supported is 2KB (value == 9).

**Accessing the DCZID_EL0**

Accesses to this register use the following encodings:

MRS <Xt>, DCZID_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2,<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HFGRTR_EL2.DCZID_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return DCZID_EL0;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRTR_EL2.DCZID_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return DCZID_EL0;
elsif PSTATE.EL == EL2 then
  return DCZID_EL0;
elsif PSTATE.EL == EL3 then
  return DCZID_EL0;

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**DISR_EL1, Deferred Interrupt Status Register**

The DISR_EL1 characteristics are:

### Purpose

Records that an SError interrupt has been consumed by an ESB instruction.

### Configuration

AArch64 System register DISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register `DISR[31:0]`. This register is present only when RAS is implemented. Otherwise, direct accesses to DISR_EL1 are **UNDEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values.

### Attributes

DISR_EL1 is a 64-bit register.

### Field descriptions

The DISR_EL1 bit assignments are:

#### When DISR_EL1.IDS == 0:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>RES0.</td>
</tr>
<tr>
<td>59</td>
<td>IDS, bit [24]: Indicates the deferred SError interrupt type.</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>A, bit [31]: Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0. This field resets to an architecturally <strong>UNKNOWN</strong> value.</td>
</tr>
<tr>
<td>30</td>
<td>Bits [30:25]: Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Bits [29:24]: Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Bits [28:23]: Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Bits [27:22]: Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Bits [26:21]: Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Bits [25:20]: Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Bits [24:19]: Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Bits [23:18]: Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Bits [22:17]: Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Bits [21:16]: Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Bits [20:15]: Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Bits [19:14]: Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Bits [18:13]: Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Bits [17:12]: Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Bits [16:11]: Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Bits [15:10]: Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Bits [14:9]: Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Bits [13:8]: Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Bits [12:7]: Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Bits [11:6]: Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Bits [10:5]: Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Bits [9:4]: Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Bits [8:3]: Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Bits [7:2]: Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Bits [6:1]: Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Bits [5:0]: Reserved, RES0.</td>
</tr>
</tbody>
</table>

---

**Table:**

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Deferred error uses architecturally-defined format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.
Bits [23:13]

Reserved, RES0.

AET, bits [12:10]

Asynchronous Error Type. See the description of ESR_ELx.AET for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort Type. See the description of ESR_ELx.EA for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]

Fault Status Code. See the description of ESR_ELx.DFSC for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

When DISR_EL1.IDS == 1:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>A</td>
</tr>
<tr>
<td>61</td>
<td>IDS</td>
</tr>
<tr>
<td>60</td>
<td>ISS</td>
</tr>
<tr>
<td>59</td>
<td>58</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>55</td>
<td>54</td>
</tr>
<tr>
<td>54</td>
<td>53</td>
</tr>
<tr>
<td>53</td>
<td>52</td>
</tr>
<tr>
<td>52</td>
<td>51</td>
</tr>
<tr>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>50</td>
<td>49</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>48</td>
<td>47</td>
</tr>
<tr>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>46</td>
<td>45</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
</tr>
<tr>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>43</td>
<td>42</td>
</tr>
<tr>
<td>42</td>
<td>41</td>
</tr>
<tr>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>40</td>
<td>39</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
</tr>
<tr>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>36</td>
<td>35</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>34</td>
<td>33</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
</tr>
</tbody>
</table>

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

This field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError interrupt type.

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Deferred error uses IMPLEMENTATION DEFINED format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

IMPLEMENTATION DEFINED.
IMPLEMENTATION DEFINED syndrome. See the description of ESR_ELx[23:0] for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

**Accessing the DISR_EL1**

An indirect write to DISR_EL1 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

DISR_EL1 is RAZ/WI if EL3 is implemented, the PE is in Non-debug state, SCR_EL3.EA == 1, and any of the following apply:

- At EL2.
- At EL1 and ((SCR_EL3.NS == 0 && SCR_EL3.EEL2 == 0) || HCR_EL2.AMO == 0).

Accesses to this register use the following encodings:

**MRS <Xt>, DISR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        VDISR_EL2 = X[t];
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    DISR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t];

**MSR DISR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        VDISR_EL2 = X[t];
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    DISR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t];
DIT, Data Independent Timing

The DIT characteristics are:

**Purpose**

Allows access to the Data Independent Timing bit.

**Configuration**

This register is present only when ARMv8.4-DIT is implemented. Otherwise, direct accesses to DIT are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DIT is a 64-bit register.

**Field descriptions**

The DIT bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| RES0| DIT | RES0| RES0|
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**Bits [63:25]**

Reserved, **RES0**.

**DIT, bit [24]**

Data Independent Timing.

<table>
<thead>
<tr>
<th>DIT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The architecture makes no statement about the timing properties of any instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>The architecture requires that:</td>
</tr>
<tr>
<td></td>
<td>• The timing of every load and store instruction is insensitive to the value of the data being loaded or stored.</td>
</tr>
<tr>
<td></td>
<td>• For certain data processing instructions, the instruction takes a time which is independent of:</td>
</tr>
<tr>
<td></td>
<td>◦ The values of the data supplied in any of its registers.</td>
</tr>
<tr>
<td></td>
<td>◦ The values of the NZCV flags.</td>
</tr>
<tr>
<td></td>
<td>• For certain data processing instructions, the response of the instruction to asynchronous exceptions does not vary based on:</td>
</tr>
<tr>
<td></td>
<td>◦ The values of the data supplied in any of its registers.</td>
</tr>
<tr>
<td></td>
<td>◦ The values of the NZCV flags.</td>
</tr>
</tbody>
</table>

The data processing instructions affected by this bit are:

• All cryptographic instructions. These instructions are:
A subset of those instructions which use the general-purpose register file. These instructions are:

- ADC, ADCS, ADD, ADDS, AND, ANDS, ASR, ASRV, BFC, BFI, BFMI, BIC, BICS, CCMN, CCMPS, CFIN, CINC, CINV, CLS, CLZ, CMP, CNEG, CSEL, CSAT, CSETM, CSI, CSINC, CSINV, CSNEG, EON, EOR, EXTR, LSL, LSLV, LSR, LSRV, MADD, MNEG, MOV, MOVK, MOVV, MOVZ, MSUB, MUL, MVN, NEG, NEGS, NGC, NGCS, NOP, ORN, ORR, RBIT, RET, REV, REV16, REV32, REV64, RMIF, ROR, RORV, SBC, SBCS, SBFIZ, SBFM, SBFX, SETF8, SETF16, SMADDL, SMNEGL, SMSUBL, SMULH, SMULL, SUB, SUBS, SXTB, SXTH, SXTW, TST, UBIFIZ, UBFM, UBFX, UMADDL, UMNEGL, UMSUBL, UMULH, UMULL, UXTH, and UXTH.

- A subset of those instructions which use the SIMD&FP register file. These instructions are:

- ABS, ADD, ADDHN, ADDHN2, ADDP, ADDY, AND, BIC, BIF, BIT, BSL, CLS, CLZ, CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMST, CNT, CRC32B, CRC32C, CRC32CH, CRC32CX, CRC32CH, CRC32CW, CRC32CX, DUP, EOR, EXT, FCSEL, INS, MLA, MLS, MOV, MOVI, MUL, MVN, MVNI, NEG, NOT, ORN, ORR, PMUL, PMULL, PMULL2, RADDHN, RADDHN2, RBIT, REV16, REV32, RSHRN, RSHRN2, RSUBHN, RSUBHN2, SABA, SABD, SABAL, SABAL2, SABDL, SABDL2, SADALP, SADDL, SADDL2, SADDP, SADDLV, SADDW, SADDW2, SHADD, SRL, SRLH, SRLH2, SHR, SHR2, SHSUB, SLI, SMAX, SMAXP, SMAXV, SMIN, SMINP, SMINV, SMLAL, SMLAL2, SMLSL, SMLSL2, SMOV, SMULL, SMULL2, SRI, SSHL, SSHLL, SSHLL2, SSRH, SSRH2, SSRA, SSUBL, SSUBL, SSUBW, SSUBW2, SUB, SUBH, SUBHN, SXTL, SXTH, TBL, TBX, TRN1, TRN2, UABA, UABAL, UABAL2, UABD, UABDL, UADLP, UADDL, UADDL2, UADDP, UADDLV, UADDW, UADDW2, UHADD, UHSUB, UMAX, UMAP, UMAXV, UMIN, UMINV, UMLAL, UMLAL2, UMLSL, UMOM, UMLSL2, UMULL, UMULL2, USHL, USHL2, USHR, USRA, USUBL, USUBL2, USUBW, USUBW2, UXTH, UXTL, UXTL2, UZP1, UZP2, XTN, XTN2, ZIP1, and ZIP2.

Note

The architecture makes no statement about the timing properties when the PSTATE.DIT bit is not set. However, it is likely that many of these instructions have timing that is invariant of the data in many situations.

In particular, Arm strongly recommends that the Armv8.3 pointer authentication instructions do not have their timing dependent on the key value used in the pointer authentication in all cases, regardless of the PSTATE.DIT bit.

This field resets to 0.

Bits [23:0]

Reserved, RES0.

Accessing the DIT

For details on the operation of the MSR (immediate) accessor, see MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  return Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL1 then
  return Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL2 then
  return Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL3 then
  return Zeros(39):PSTATE.DIT:Zeros(24);
MSR DIT, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
PSTATE.DIT = X[t]<24>;
elsif PSTATE.EL == EL1 then
PSTATE.DIT = X[t]<24>;
elsif PSTATE.EL == EL2 then
PSTATE.DIT = X[t]<24>;
elsif PSTATE.EL == EL3 then
PSTATE.DIT = X[t]<24>;

MSR DIT, #<imm>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b010</td>
</tr>
</tbody>
</table>

27/09/2019 18:22; 6134483bd14dc8c12a99c984c8be3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DLR_EL0, Debug Link Register

The DLR_EL0 characteristics are:

**Purpose**

In Debug state, holds the address to restart from.

**Configuration**

AArch64 System register DLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DLR[31:0].

**Attributes**

DLR_EL0 is a 64-bit register.

**Field descriptions**

The DLR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Restart address</td>
</tr>
<tr>
<td>31</td>
<td>Restart address</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Restart address.

**Accessing the DLR_EL0**

Accesses to this register use the following encodings:

\[
\text{MRS} \ <Xt>, \ \text{DLR_EL0}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if !Halted() then
    UNDEFINED;
else
    return DLR_EL0;

\[
\text{MSR} \ \text{DLR_EL0}, \ <Xt>
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if !Halted() then
    UNDEFINED;
else
    DLR_EL0 = X[t];
DSPSR_EL0, Debug Saved Program Status Register

The DSPSR_EL0 characteristics are:

**Purpose**

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

**Configuration**

AArch64 System register DSPSR_EL0 bits [31:0] are architecturally mapped to AArch32 System register DSPSR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DSPSR_EL0 is a 64-bit register.

**Field descriptions**

The DSPSR_EL0 bit assignments are:

When exiting Debug state to AArch32 state:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Copied to PSTATE.N on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Copied to PSTATE.Z on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Copied to PSTATE.C on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

**V, bit [28]**

Overflow Condition flag. Copied to PSTATE.V on exiting Debug state.
This field resets to an architecturally **UNKNOWN** value.

**Q, bit [27]**

Overflow or saturation flag. Copied to PSTATE.Q on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Copied to PSTATE.IT[1:0] on exiting Debug state.

On exiting Debug state DSPSR_EL0.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**DIT, bit [24]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Copied to PSTATE.DIT on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**SSBS, bit [23]**

*When ARMv8.0-SSBS is implemented:*

Speculative Store Bypass. Copied to PSTATE.SSBS on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**PAN, bit [22]**

*When ARMv8.1-PAN is implemented:*

Privileged Access Never. Copied to PSTATE.PAN on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**SS, bit [21]**

Software Step. Copied to PSTATE.SS on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.
IL, bit [20]

Illegal Execution state. Copied to PSTATE.I on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Copied to PSTATE.GE on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

IT[7:2], bits [15:10]

If-Then. Copied to PSTATE.IT[7:2] on exiting Debug state.

DSPSR_EL0.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR_EL0.E is RES0. If the implementation does not support little-endian operation, DSPSR_EL0.E is RES1. On exiting Debug state, if the implementation does not support big-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Copied to PSTATE.A on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Copied to PSTATE.I on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Copied to PSTATE.F on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Copied to PSTATE.T on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Copied to PSTATE.nRW on exiting Debug state.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>AArch32 execution state.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally unknown value.

M[3:0], bits [3:0]

AArch32 Mode. Copied to PSTATE.M[3:0] on exiting Debug state.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>User.</td>
</tr>
<tr>
<td>0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0100</td>
<td>Monitor.</td>
</tr>
<tr>
<td>0101</td>
<td>Abort.</td>
</tr>
<tr>
<td>0110</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0111</td>
<td>Undefined.</td>
</tr>
<tr>
<td>1000</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If DSPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in ‘Illegal return events from AArch64 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally unknown value.

When entering Debug state from AArch64 state and exiting Debug state to AArch64 state:

<table>
<thead>
<tr>
<th>Bits [63:32]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>59 58 57 56</td>
<td></td>
</tr>
<tr>
<td>55 54 53 52</td>
<td></td>
</tr>
<tr>
<td>51 50 49 48</td>
<td></td>
</tr>
<tr>
<td>47 46 45 44</td>
<td></td>
</tr>
<tr>
<td>43 42 41 40</td>
<td></td>
</tr>
<tr>
<td>39 38 37 36</td>
<td></td>
</tr>
<tr>
<td>35 34 33 32</td>
<td></td>
</tr>
<tr>
<td>31 30 29 28</td>
<td></td>
</tr>
<tr>
<td>27 26 25 24</td>
<td></td>
</tr>
<tr>
<td>23 22 21 20</td>
<td></td>
</tr>
<tr>
<td>19 18 17 16</td>
<td></td>
</tr>
<tr>
<td>15 14 13 12</td>
<td></td>
</tr>
<tr>
<td>11 10  9  8</td>
<td></td>
</tr>
<tr>
<td>7  6  5  4</td>
<td></td>
</tr>
<tr>
<td>3  2  1  0</td>
<td></td>
</tr>
</tbody>
</table>

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

This field resets to an architecturally unknown value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

This field resets to an architecturally unknown value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

This field resets to an architecturally unknown value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

This field resets to an architecturally unknown value.
Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When ARMv8.5-MemTag is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on entering Debug state, and copied to PSTATE.TCO on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When ARMv8.2-UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on entering Debug state, and copied to PSTATE.UAO on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.
This field resets to an architecturally unknown value.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

This field resets to an architecturally unknown value.

**Bits [19:13]**

Reserved, RES0.

**SSBS, bit [12]**

*When ARMv8.0-SSBS is implemented:*

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

This field resets to an architecturally unknown value.

*Otherwise:*

Reserved, RES0.

**BTYPE, bits [11:10]**

*When ARMv8.5-BTI is implemented:*

Branch Type Indicator. Set to the value of PSTATE.BTYPE on entering Debug state, and copied to PSTATE.BTYPE on exiting Debug state.

This field resets to an architecturally unknown value.

*Otherwise:*

Reserved, RES0.

**D, bit [9]**

Debug exception mask. Set to the value of PSTATE.D on entering Debug state, and copied to PSTATE.D on exiting Debug state.

This field resets to an architecturally unknown value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

This field resets to an architecturally unknown value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

This field resets to an architecturally unknown value.
**F, bit [6]**

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**Bit [5]**

Reserved, RES0.

**M[4], bit [4]**

Execution state. Set to 0b0, the value of PSTATE.nRW, on entering Debug state from AArch64 state, and copied to PSTATE.nRW on exiting Debug state.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch64 execution state.</td>
</tr>
</tbody>
</table>

If AArch32 is not supported at any Exception level, this bit is RES0.

This field resets to an architecturally **UNKNOWN** value.

**M[3:0], bits [3:0]**

AArch64 Exception level and selected Stack Pointer.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL0t</td>
</tr>
<tr>
<td>0b0100</td>
<td>EL1t</td>
</tr>
<tr>
<td>0b0101</td>
<td>EL1h</td>
</tr>
<tr>
<td>0b1000</td>
<td>EL2t</td>
</tr>
<tr>
<td>0b1001</td>
<td>EL2h</td>
</tr>
<tr>
<td>0b1100</td>
<td>EL3t</td>
</tr>
<tr>
<td>0b1101</td>
<td>EL3h</td>
</tr>
</tbody>
</table>

Other values are reserved. If DSPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on entering Debug state and copied to PSTATE.EL on exiting Debug state.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on entering Debug state and copied to PSTATE.SP on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the DSPSR_EL0**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS &lt;Xt&gt;, DSPSR_EL0</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
</tr>
<tr>
<td>0b11</td>
</tr>
</tbody>
</table>

if !Halted() then
   UNDEFINED;
else
   return DSPSR_EL0;
MSR DSPSR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if !Halted() then
  UNDEFINED;
else
  DSPSR_EL0 = X[t];
DVP RCTX, Data Value Prediction Restriction by Context

The DVP RCTX characteristics are:

**Purpose**

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, data value prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

**Note**

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

**Configuration**

This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to DVP RCTX are UNDEFINED.

**Attributes**

DVP RCTX is a 64-bit System instruction.

**Field descriptions**

The DVP RCTX input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:49]</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>[48]</td>
<td>GVMID</td>
</tr>
<tr>
<td>[47:32]</td>
<td>VMID</td>
</tr>
<tr>
<td>[31:16]</td>
<td>ASID</td>
</tr>
<tr>
<td>[15:0]</td>
<td>GASID</td>
</tr>
</tbody>
</table>

**Bits [63:49]**

Reserved, RES0.

**GVMID, bit [48]**

Execution of this instruction applies to all VMIDs or a specified VMID.
### GVMID

<table>
<thead>
<tr>
<th>GVMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

### VMID, bits [47:32]

Only applies when bit[48] is 0 and one of:

- an EL1 context.
- an EL0 context when \(\text{HCR\_EL2}.E2H==0\) or \(\text{HCR\_EL2}.TGE==0\).

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and \(\text{HCR\_EL2}.E2H==0\) or \(\text{HCR\_EL2}.TGE==0\) then this field is treated as the current VMID.

When the instruction is executed at EL0 and \(\text{HCR\_EL2}.E2H==1\) and \(\text{HCR\_EL2}.TGE==1\) then this field is ignored.

### Bits [31:27]

Reserved, RES0.

### NS, bit [26]

Security State.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state.</td>
</tr>
</tbody>
</table>

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

### EL, bits [25:24]

Exception Level

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>EL0.</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>EL3.</td>
</tr>
</tbody>
</table>

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

### Bits [23:17]

Reserved, RES0.

### GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

<table>
<thead>
<tr>
<th>GASID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>
If the instruction is executed at EL0, then this field has an Effective value of 0.

**ASID, bits [15:0]**

Only applies for an EL0 context and when bit[16] is 0.
Otherwise this field is RES0.
When the instruction is executed at EL0 then this field is treated as the current ASID.

**Executing the DVP RCTX instruction**

Accesses to this instruction use the following encodings:

DVP RCTX, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
else
AArch64.SystemAccessTrap(EL1, 0x18);
elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DVPRCTX == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && SCTLR_EL2.EnRCTX == '0' then
AArch64.SystemAccessTrap(EL2, 0x18);
else
DVP_RCTX(X[t]);
elseif PSTATE.EL == EL1 then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DVPRCTX == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
else
DVP_RCTX(X[t]);
elseif PSTATE.EL == EL2 then
DVP_RCTX(X[t]);
elseif PSTATE.EL == EL3 then
DVP_RCTX(X[t]);
ELR_EL1, Exception Link Register (EL1)

The ELR_EL1 characteristics are:

**Purpose**

When taking an exception to EL1, holds the address to return to.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ELR_EL1 is a 64-bit register.

**Field descriptions**

The ELR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Return address</td>
</tr>
<tr>
<td>62</td>
<td>Return address</td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Return address.

An exception return from EL1 using AArch64 makes ELR_EL1 become UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**Accessing the ELR_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic ELR_EL1 or ELR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x230];
  else
    return ELR_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return ELR_EL2;
  else
    return ELR_EL1;
elsif PSTATE.EL == EL3 then
  ELR_EL1 = X[t];
else
  UNDEFINED;
endif

MSR ELR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x230] = X[t];
  else
    ELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    ELR_EL2 = X[t];
  else
    ELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  ELR_EL1 = X[t];
else
  UNDEFINED;
endif

MRS <Xt>, ELR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x230];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return ELR_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return ELR_EL1;
  else
    UNDEFINED;
else
  UNDEFINED;
endif
if PSTATE.EL == EL0 then
UNCHANGED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x230] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    ELR_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    ELR_EL1 = X[t];
  else
    UNDEFINED;

MRS <Xt>, ELR_EL2

if PSTATE.EL == EL0 then
UNCHANGED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '11' then
    ELR_EL1 = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ELR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  ELR_EL2 = X[t];
ELR_EL2, Exception Link Register (EL2)

The ELR_EL2 characteristics are:

**Purpose**

When taking an exception to EL2, holds the address to return to.

**Configuration**

AArch64 System register ELR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ELR_hyp[31:0].

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally unknown values.

**Attributes**

ELR_EL2 is a 64-bit register.

**Field descriptions**

The ELR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Return address | Return address |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:0]**

Return address.

An exception return from EL2 using AArch64 makes ELR_EL2 become unknown.

When EL2 is in AArch32 Execution state and an exception is taken from EL0, EL1, or EL2 to EL3 and AArch64 execution, the upper 32-bits of ELR_EL2 are either set to 0 or hold the same value that they did before AArch32 execution. Which option is adopted is determined by an implementation, and might vary dynamically within an implementation. Correspondingly software must regard the value as being an unknown choice between the two values.

This field resets to an architecturally unknown value.

**Accessing the ELR_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic ELR_EL2 or ELR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ELR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    return ELR_EL2;
elsif PSTATE.EL == EL3 then
    return ELR_EL2;
endif

MSR ELR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ELR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    ELR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t];
endif

MRS <Xt>, ELR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x230];
    else
        return ELR_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ELR_EL2;
    else
        return ELR_EL1;
    endif
elsif PSTATE.EL == EL3 then
    return ELR_EL1;
endif

MSR ELR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x230] = X[t];
  else
    ELR_EL1 = X[t];
  end
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    ELR_EL2 = X[t];
  else
    ELR_EL1 = X[t];
  end
elsif PSTATE.EL == EL3 then
  ELR_EL1 = X[t];
ELR_EL3, Exception Link Register (EL3)

The ELR_EL3 characteristics are:

**Purpose**

When taking an exception to EL3, holds the address to return to.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to ELR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ELR_EL3 is a 64-bit register.

**Field descriptions**

The ELR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Return address | Return address |

**Bits [63:0]**

Return address.

An exception return from EL3 using AArch64 makes ELR_EL3 become UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**Accessing the ELR_EL3**

Accesses to this register use the following encodings:

```
MRS <Xt>, ELR_EL3
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ELR_EL3;
MSR ELR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ELR_EL3 = X[t];
**ERRIDR_EL1, Error Record ID Register**

The ERRIDR_EL1 characteristics are:

**Purpose**

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

**Configuration**

AArch64 System register ERRIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERRIDR[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRIDR_EL1 are **UNDEFINED**.

**Attributes**

ERRIDR_EL1 is a 64-bit register.

**Field descriptions**

The ERRIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-16</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>15-0</td>
<td>Highest numbered index of the records that can be accessed through the Error Record System registers plus one. Zero indicates that no records can be accessed through the Error Record System registers. Each implemented record is owned by a node. A node might own multiple records.</td>
</tr>
</tbody>
</table>

**Accessing the ERRIDR_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ERRIDR_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRT_EL2.ERRIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ERRIDR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERRIDR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return ERRIDR_EL1;
end
ERRSELR_EL1, Error Record Select Register

The ERRSELR_EL1 characteristics are:

**Purpose**

Selects an error record to be accessed through the Error Record System registers.

**Configuration**

AArch64 System register ERRSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERRSELR[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRSELR_EL1 are UNDEFINED.

If ERRIDR_EL1 indicates that zero records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR_EL1 is UNDEFINED or RES0.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ERRSELR_EL1 is a 64-bit register.

**Field descriptions**

The ERRSELR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | SEL |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:16]**

Reserved, RES0.

**SEL, bits [15:0]**

Selects the record accessed through the ERX registers.

For example, if ERRSELR_EL1.SEL is set to 0x4, then direct reads and writes of ERXSTATUS_EL1 access ERR4STATUS.

If ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then all of the following apply:

- The value read back from ERRSELR_EL1.SEL is UNKNOWN.
- One of the following occurs:
  - An UNKNOWN error record is selected.
  - The ERX* registers are RAZ/WI.
  - ERX* register reads and writes are NOPs.
  - ERX* register reads and writes are UNDEFINED.

This field resets to an architecturally UNKNOWN value.
Accessing the **ERRSELR_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, ERRSELR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HFGWTR_EL2.ERRSELR_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return ERRSELR_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      ERRSELR_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    return ERRSELR_EL1;
else
    return ERRSELR_EL1;
endif

**MSR ERRSELR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return ERRSELR_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      ERRSELR_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    return ERRSELR_EL1;
else
    return ERRSELR_EL1;
endif
ERXADDR_EL1, Selected Error Record Address Register

The ERXADDR_EL1 characteristics are:

**Purpose**

Accesses \( \text{ERR}_n\text{ADDR} \) for the error record selected by \( \text{ERRSELR_EL1}.\text{SEL} \).

**Configuration**

AArch64 System register ERXADDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register \( \text{ERXADDR}[31:0] \).

AArch64 System register ERXADDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register \( \text{ERXADDR2}[31:0] \).

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXADDR_EL1 are UNDEFINED.

**Attributes**

ERXADDR_EL1 is a 64-bit register.

**Field descriptions**

The ERXADDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ERR(_n)ADDR</td>
</tr>
<tr>
<td>62</td>
<td>ERR(_n)ADDR</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>ERR(_n)ADDR</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

ERXADDR_EL1 accesses \( \text{ERR}_n\text{ADDR} \), where \( n \) is the value in \( \text{ERRSELR_EL1}.\text{SEL} \).

**Accessing the ERXADDR_EL1**

If \( \text{ERRIDR_EL1}.\text{NUM} == 0 \) or \( \text{ERRSELR_EL1}.\text{SEL} \) is set to a value greater than or equal to \( \text{ERRIDR_EL1}.\text{NUM} \), then one of the following occurs:

- An UNKNOWN record is selected.
- ERXADDR_EL1 is RAZ/WI.
- Direct reads and writes of ERXADDR_EL1 are NOPs.
- Direct reads and writes of ERXADDR_EL1 are UNDEFINED.

Accesses to this register use the following encodings:

\[
\begin{array}{cccccc}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b11 & 0b000 & 0b0101 & 0b0100 & 0b011 \\
\end{array}
\]
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.ERXADDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXADDR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXADDR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return ERXADDR_EL1;
end

MSR ERXADDR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.ERXADDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXADDR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXADDR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    ERXADDR_EL1 = X[t];
ERXCTRL_EL1, Selected Error Record Control Register

The ERXCTRL_EL1 characteristics are:

**Purpose**

Accesses ERR<n>CTRL for the error record selected by ERRSELR_EL1.SEL.

**Configuration**

AArch64 System register ERXCTRL_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXCTRL[31:0].

AArch64 System register ERXCTRL_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXCTRL2[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXCTRL_EL1 are **UNDEFINED**.

**Attributes**

ERXCTRL_EL1 is a 64-bit register.

**Field descriptions**

The ERXCTRL_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>ERR&lt;n&gt;CTRL</td>
</tr>
<tr>
<td>31-0</td>
<td>ERR&lt;n&gt;CTRL</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

ERXCTRL_EL1 accesses ERR<n>CTRL, where n is the value in ERRSELR_EL1.SEL.

**Accessing the ERXCTRL_EL1**

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXCTRL_EL1 is RAZ/WI.
- Direct reads and writes of ERXCTRL_EL1 are NOPs.
- Direct reads and writes of ERXCTRL_EL1 are **UNDEFINED**.

Accesses to this register use the following encodings:

MRS <Xt>, ERXCTRL_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>000</td>
<td>00101</td>
<td>00100</td>
<td>001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.ERXCTLR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ERXCTLR_EL1;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXCTLR_EL1;
  endif
elsif PSTATE.EL == EL3 then
  return ERXCTLR_EL1;
endif

MSR ERXCTLR_EL1, <Xt>
**ERXFR_EL1, Selected Error Record Feature Register**

The ERXFR_EL1 characteristics are:

### Purpose

Accesses **ERR<n>FR** for the error record selected by **ERRSELR_EL1.SEL**.

### Configuration

AArch64 System register ERXFR_EL1 bits [31:0] are architecturally mapped to AArch32 System register **ERXFR[31:0]**.

AArch64 System register ERXFR_EL1 bits [63:32] are architecturally mapped to AArch32 System register **ERXFR2[31:0]**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXFR_EL1 are UNDEFINED.

### Attributes

ERXFR_EL1 is a 64-bit register.

### Field descriptions

The ERXFR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>ERXFR_EL1 accesses <strong>ERR&lt;n&gt;FR</strong>, where n is the value in <strong>ERRSELR_EL1.SEL</strong>.</th>
</tr>
</thead>
</table>

### Accessing the ERXFR_EL1

If **ERRIDR_EL1.NUM == 0** or **ERRSELR_EL1.SEL** is set to a value greater than or equal to **ERRIDR_EL1.NUM**, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXFR_EL1 is RAZ.
- Direct reads of ERXFR_EL1 are NOPs.
- Direct reads of ERXFR_EL1 are UNDEFINED.

Accesses to this register use the following encodings:

MRS <Xt>, ERXFR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HGRTR_EL2.ERXFR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXFR_EL1;
  end if
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXFR_EL1;
  end if
elsif PSTATE.EL == EL3 then
  return ERXFR_EL1;
end if
The ERXMISC0_EL1 characteristics are:

**Purpose**

Accesses ERR<n>MISC0 for the error record selected by ERXSELR_EL1.SEL.

**Configuration**

AArch64 System register ERXMISC0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC0[31:0].

AArch64 System register ERXMISC0_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC1[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXMISC0_EL1 are UNDEFINED.

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

**Note**

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.

**Attributes**

ERXMISC0_EL1 is a 64-bit register.

**Field descriptions**

The ERXMISC0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ERR&lt;n&gt;MISC0</td>
</tr>
<tr>
<td>62</td>
<td>ERR&lt;n&gt;MISC0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>ERR&lt;n&gt;MISC0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

ERXMISC0_EL1 accesses ERR<n>MISC0, where n is the value in ERXSELR_EL1.SEL.
Accessing the ERXMISC0_EL1

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC0_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC0_EL1 are NOPs.
- Direct reads and writes of ERXMISC0_EL1 are **UNDEFINED**.

Accesses to this register use the following encodings:

**MRS <Xt>, ERXMISC0_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsisf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsisf EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXMISCn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsisf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXMISC0_EL1;
elsisf PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ERXMISC0_EL1;
elsisf PSTATE.EL == EL3 then
  return ERXMISC0_EL1;
```

**MSR ERXMISC0_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsisf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsisf EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXMISCn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsisf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  ERXMISC0_EL1 = X[t];
elsisf PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  ERXMISC0_EL1 = X[t];
elsisf PSTATE.EL == EL3 then
  ERXMISC0_EL1 = X[t];
```
ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1

The ERXMISC1_EL1 characteristics are:

**Purpose**

Accesses \( \text{ERR}<n>\text{MISC}1 \) for the error record selected by \( \text{ERRSELR_EL1}.\text{SEL} \).

**Configuration**

AArch64 System register ERXMISC1_EL1 bits [31:0] are architecturally mapped to AArch32 System register \( \text{ERXMISC2}[31:0] \).

AArch64 System register ERXMISC1_EL1 bits [63:32] are architecturally mapped to AArch32 System register \( \text{ERXMISC3}[31:0] \).

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXMISC1_EL1 are undefined.

For implementation defined fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any implementation defined syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

**Note**

Arm recommends that any implementation defined syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or implementation defined, interrupt request are disabled at Cold reset and are enabled by software writing an implementation defined non-zero value to an implementation defined field in \( \text{ERR}<q>\text{CTRL} \), where \( q \) is the index of the first error record owned by the same node as error record \( n \). If the node owns a single record then \( q = n \).

**Attributes**

ERXMISC1_EL1 is a 64-bit register.

**Field descriptions**

The ERXMISC1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Index</th>
<th>Bit Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>( \text{ERR}&lt;n&gt;\text{MISC}1 )</td>
</tr>
<tr>
<td>62</td>
<td>( \text{ERR}&lt;n&gt;\text{MISC}1 )</td>
</tr>
<tr>
<td>31</td>
<td>( \text{ERR}&lt;n&gt;\text{MISC}1 )</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

ERXMISC1_EL1 accesses \( \text{ERR}<n>\text{MISC}1 \), where \( n \) is the value in \( \text{ERRSELR_EL1}.\text{SEL} \).
Accessing the ERXMISC1_EL1

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC1_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC1_EL1 are NOPs.
- Direct reads and writes of ERXMISC1_EL1 are **UNDEFINED**.

Accesses to this register use the following encodings:

**MRS <Xt>, ERXMISC1_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ERXMISCn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXMISC1_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return ERXMISC1_EL1;
  elsif PSTATE.EL == EL3 then
    return ERXMISC1_EL1;
endif

**MSR ERXMISC1_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ERXMISCn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXMISC1_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      ERXMISC1_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    ERXMISC1_EL1 = X[t];
The ERXMISC2_EL1 characteristics are:

### Purpose

Accesses ERR<n>MISC2 for the error record selected by ERRSELR_EL1.SEL.

### Configuration

AArch64 System register ERXMISC2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC4[31:0].

AArch64 System register ERXMISC2_EL1 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC5[31:0].

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXMISC2_EL1 are UNDEFINED.

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

---

### Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.

### Attributes

ERXMISC2_EL1 is a 64-bit register.

### Field descriptions

The ERXMISC2_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>62</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>61</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>60</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>59</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>58</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>57</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>56</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>55</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>54</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>53</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>52</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>51</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>50</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>49</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>48</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>47</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>46</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>45</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>44</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>43</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>42</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>41</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>40</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>39</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>38</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>37</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>36</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>35</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>34</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>33</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
<tr>
<td>32</td>
<td>ERR&lt;n&gt;MISC2</td>
</tr>
</tbody>
</table>

### Bits [63:0]

ERXMISC2_EL1 accesses ERR<n>MISC2, where n is the value in ERRSELR_EL1.SEL.
Accessing the ERXMISC2_EL1

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC2_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC2_EL1 are NOPs.
- Direct reads and writes of ERXMISC2_EL1 are **UNDEFINED**.

Accesses to this register use the following encodings:

**MRS <Xt>, ERXMISC2_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXMISC2_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXMISC2_EL1;
    endif
elsif PSTATE.EL == EL3 then
    return ERXMISC2_EL1;
endif

**MSR ERXMISC2_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.ERXMISCn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC2_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ERXMISC2_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    ERXMISC2_EL1 = X[t];
**ERXMISC3_EL1, Selected Error Record Miscellaneous Register 3**

The ERXMISC3_EL1 characteristics are:

**Purpose**

Accesses \texttt{ERR<n>MISC3} for the error record selected by \texttt{ERRSELR_EL1.SEL}.

**Configuration**

AArch64 System register ERXMISC3_EL1 bits [31:0] are architecturally mapped to AArch32 System register \texttt{ERXMISC6[31:0]}.  

AArch64 System register ERXMISC3_EL1 bits [63:32] are architecturally mapped to AArch32 System register \texttt{ERXMISC7[31:0]}.  

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXMISC3_EL1 are UNDEFINED.

For \texttt{IMPLEMENTATION DEFINED} fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any \texttt{IMPLEMENTATION DEFINED} syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

---

**Note**

Arm recommends that any \texttt{IMPLEMENTATION DEFINED} syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or \texttt{IMPLEMENTATION DEFINED}, interrupt request are disabled at Cold reset and are enabled by software writing a \texttt{IMPLEMENTATION DEFINED} non-zero value to an \texttt{IMPLEMENTATION DEFINED} field in \texttt{ERR<q>CTRL}, where \( q \) is the index of the first error record owned by the same node as error record \( n \). If the node owns a single record then \( q = n \).

---

**Attributes**

ERXMISC3_EL1 is a 64-bit register.

**Field descriptions**

The ERXMISC3_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>\texttt{ERR&lt;n&gt;MISC3}</td>
</tr>
<tr>
<td>31</td>
<td>\texttt{ERR&lt;n&gt;MISC3}</td>
</tr>
<tr>
<td>0</td>
<td>\texttt{ERR&lt;n&gt;MISC3}</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

ERXMISC3_EL1 accesses \texttt{ERR<n>MISC3}, where \( n \) is the value in \texttt{ERRSELR_EL1.SEL}. 

---

*Page 447*
Accessing the ERXMISC3_EL1

If $\text{ERRIDR_EL1}.\text{NUM} == 0$ or $\text{ERRSELR_EL1}.\text{SEL}$ is set to a value greater than or equal to $\text{ERRIDR_EL1}.\text{NUM}$, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC3_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC3_EL1 are NOPs.
- Direct reads and writes of ERXMISC3_EL1 are **UNDEFINED**.

Accesses to this register use the following encodings:

MRS $<\text{Xt}>, \text{ERXMISC3_EL1}$

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elses PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.ERXMISCn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXMISC3_EL1;
  endelsif
elses PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXMISC3_EL1;
  endelsif
elses PSTATE.EL == EL3 then
  return ERXMISC3_EL1;
endelsif

MSR ERXMISC3_EL1, $<\text{Xt}>$

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elses PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ERXMISCn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXMISC3_EL1 = X[t];
  endelsif
elses PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXMISC3_EL1 = X[t];
  endelsif
elses PSTATE.EL == EL3 then
  ERXMISC3_EL1 = X[t];
endelsif
ERXPFGCDN_EL1, Selected Pseudo-fault Generation Countdown Register

The ERXPFGCDN_EL1 characteristics are:

### Purpose

Accesses the ERR<n>PFGCDN register for the error record selected by ERRSELR_EL1.SEL.

### Configuration

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXPFGCDN_EL1 are UNDEFINED.

### Attributes

ERXPFGCDN_EL1 is a 64-bit register.

### Field descriptions

The ERXPFGCDN_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ERR<n>PFGCDN**

**ERR<n>PFGCDN**

### Bits [63:0]

ERXPFGCDN_EL1 accesses ERR<n>PFGCDN, where n is the value in ERRSELR_EL1.SEL.

### Accessing the ERXPFGCDN_EL1

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record that does not implement the RAS Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

### Note

An error record does not implement the RAS Common Fault Injection Model Extension when ERR<n>FR.INJ == 0b00.
Accesses to this register use the following encodings:

MRS <Xt>, ERXPFGCDN_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FIEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ERXPFGCDN_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXPFGCDN_EL1;
  endif;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXPFGCDN_EL1;
  endif;
elsif PSTATE.EL == EL3 then
  return ERXPFGCDN_EL1;
endif;

MSR ERXPFGCDN_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FIEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ERXPFGCDN_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXPFGCDN_EL1 = X[t];
  endif;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXPFGCDN_EL1 = X[t];
  endif;
elsif PSTATE.EL == EL3 then
  ERXPFGCDN_EL1 = X[t];
ERXPFGCTL_EL1, Selected Pseudo-fault Generation Control Register

The ERXPFGCTL_EL1 characteristics are:

**Purpose**

Accesses the ERR<n>PFGCTL register for the error record selected by ERRSELR_EL1.SEL.

**Configuration**

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXPFGCTL_EL1 are UNDEFINED.

**Attributes**

ERXPFGCTL_EL1 is a 64-bit register.

**Field descriptions**

The ERXPFGCTL_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

>> |

**Bits [63:0]**

ERXPFGCTL_EL1 accesses ERR<n>PFGCTL, where n is the value in ERRSELR_EL1.SEL.

**Accessing the ERXPFGCTL_EL1**

If ERRIDR_EL1.NUM == 0 or ERRSELR_EL1.SEL is set to a value greater than or equal to ERRIDR_EL1.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

If ERRSELR_EL1.SEL selects an error record that does not implement the RAS Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

**Note**

An error record does not implement the RAS Common Fault Injection Model Extension when ERR<n>FR.INJ == 0b00.
Accesses to this register use the following encodings:

MRS <Xt>, ERXPFGCTL_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FIEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXPFGCTL_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ERXPFGCTL_EL1;
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXPFGCTL_EL1;
  end if;
elsif PSTATE.EL == EL3 then
  return ERXPFGCTL_EL1;
else
  ERXPFGCTL_EL1 = X[t];
end if;

MSR ERXPFGCTL_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FIEN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXPFGCTL_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    ERXPFGCTL_EL1 = X[t];
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIEN == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXPFGCTL_EL1 = X[t];
  end if;
elsif PSTATE.EL == EL3 then
  ERXPFGCTL_EL1 = X[t];
else
  ERXPFGCTL_EL1 = X[t];
end if;
The ERXPFGF_EL1 characteristics are:

**Purpose**

Accesses the $\text{ERR<n>PFGF}$ register for the error record selected by $\text{ERRSELR_EL1}.\text{SEL}$.

**Configuration**

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXPFGF_EL1 are UNDEFINED.

**Attributes**

ERXPFGF_EL1 is a 64-bit register.

**Field descriptions**

The ERXPFGF_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| \text{ERR<n>PFGF} | \text{ERR<n>PFGF} |

**Bits [63:0]**

ERXPFGF_EL1 accesses $\text{ERR<n>PFGF}$, where $n$ is the value in $\text{ERRSELR_EL1}.\text{SEL}$.

**Accessing the ERXPFGF_EL1**

If $\text{ERRIDR_EL1}.\text{NUM} == 0$ or $\text{ERRSELR_EL1}.\text{SEL}$ is set to a value greater than or equal to $\text{ERRIDR_EL1}.\text{NUM}$, then one of the following occurs:

- An $\text{UNKNOWN}$ record is selected.
- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

If $\text{ERRSELR_EL1}.\text{SEL}$ selects an error record that does not implement the RAS Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

**Note**

An error record does not implement the RAS Common Fault Injection Model Extension when $\text{ERR<n>FR}.\text{INJ} == 0b00$. 

---

Page 454
Accesses to this register use the following encodings:

\[
\text{MRS }<\text{Xt}, \text{ ERXPFGF_EL1} \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if \( \text{PSTATE.EL} == \text{EL0} \) then
    UNDEFINED;
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
    if \( \text{EL2Enabled}() \&\& !\text{ELUsingAArch32(EL2)} \&\& \text{HCR_EL2.FIEN} == '0' \) then
        \text{AArch64.SystemAccessTrap(EL2, 0x18)};
    else
        \text{ERXPFGF_EL1 == '1'} \&\& \text{HGRTR_EL2.ERXPFGF_EL1 == '1'} \&\& \text{HaveEL(EL3) || SCR_EL3.FGTEn == '1'} \&\& \text{AArch64.SystemAccessTrap(EL3, 0x18)};
    else
        return \text{ERXPFGF_EL1};
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
    if \( \text{HaveEL(EL3)} \&\& !\text{ELUsingAArch32(EL3)} \&\& \text{SCR_EL3.FIEN} == '0' \) then
        \text{AArch64.SystemAccessTrap(EL3, 0x18)};
    else
        return \text{ERXPFGF_EL1};
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
    return \text{ERXPFGF_EL1};
ERXSTATUS_EL1, Selected Error Record Primary Status Register

The ERXSTATUS_EL1 characteristics are:

**Purpose**

Accesses `ERR<n>STATUS` for the error record selected by `ERRSELR_EL1.SEL`.

**Configuration**

AArch64 System register ERXSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register `ERXSTATUS[31:0]`.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXSTATUS_EL1 are **UNDEFINED**.

For **IMPLEMENTATION DEFINED** fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any **IMPLEMENTATION DEFINED** syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

**Note**

Arm recommends that any **IMPLEMENTATION DEFINED** syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or **IMPLEMENTATION DEFINED** interrupt request are disabled at Cold reset and are enabled by software writing a **IMPLEMENTATION DEFINED** non-zero value to an **IMPLEMENTATION DEFINED** field in `ERR<q>CTRL`, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.

**Attributes**

ERXSTATUS_EL1 is a 64-bit register.

**Field descriptions**

The ERXSTATUS_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits [63:0]</td>
</tr>
</tbody>
</table>

ERXSTATUS_EL1 accesses `ERR<n>STATUS`, where n is the value in `ERRSELR_EL1.SEL`.

**Accessing the ERXSTATUS_EL1**

If `ERRIDR_EL1.NUM == 0` or `ERRSELR_EL1.SEL` is set to a value greater than or equal to `ERRIDR_EL1.NUM`, then one of the following occurs:

- An unknown record is selected.
• ERXSTATUS_EL1 is RAZ/WI.
• Direct reads and writes of ERXSTATUS_EL1 are NOPs.
• Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED.

Accesses to this register use the following encodings:

MRS <Xt>, ERXSTATUS_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.ERXSTATUS_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ERXSTATUS_EL1;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXSTATUS_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  ERXSTATUS_EL1 = X[t];
else
  return ERXSTATUS_EL1;
endif

MSR ERXSTATUS_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWR_EL2.ERXSTATUS_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXSTATUS_EL1 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ERXSTATUS_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  ERXSTATUS_EL1 = X[t];
else
  return ERXSTATUS_EL1;
endif
ESR_EL1, Exception Syndrome Register (EL1)

The ESR_EL1 characteristics are:

**Purpose**

Holds syndrome information for an exception taken to EL1.

**Configuration**

AArch64 System register ESR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DFSR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ESR_EL1 is a 64-bit register.

**Field descriptions**

The ESR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>62</td>
<td>Exception Class. Indicates the reason for the exception that this register holds information about.</td>
</tr>
<tr>
<td>61</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>60</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>59</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>58</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>57</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>56</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>55</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>54</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>53</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>52</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>51</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>50</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>49</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>48</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>47</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>46</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>45</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>44</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>43</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>42</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>41</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>40</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>39</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>38</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>37</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>36</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>35</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>34</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>33</td>
<td>EC, bits [31:26]</td>
</tr>
<tr>
<td>32</td>
<td>EC, bits [31:26]</td>
</tr>
</tbody>
</table>

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL1, the value of ESR_EL1 is UNKNOWN. The value written to ESR_EL1 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

**Bits [63:32]**

Reserved, RES0.

**EC, bits [31:26]**

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:
<table>
<thead>
<tr>
<th>EC</th>
<th>Meaning</th>
<th>ISS</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Unknown reason.</td>
<td>ISS encoding for exceptions with an unknown reason</td>
<td></td>
</tr>
<tr>
<td>0b000001</td>
<td>Trapped WFI or WFE instruction execution.</td>
<td>ISS encoding for an exception from a WFI or WFE instruction</td>
<td>Conditional WFE and WFI instructions that fail their condition code check do not cause an exception.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for an exception from an MCR or MRC access</td>
<td></td>
</tr>
<tr>
<td>0b000100</td>
<td>Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for an exception from an MCRR or MRRC access</td>
<td></td>
</tr>
<tr>
<td>0b000101</td>
<td>Trapped MCR or MRC access with (coproc==0b1110).</td>
<td>ISS encoding for an exception from an MCR or MRC access</td>
<td></td>
</tr>
<tr>
<td>0b000110</td>
<td>Trapped LDC or STC access.</td>
<td>ISS encoding for an exception from an LDC or STC instruction</td>
<td>The only architected uses of these instruction are:</td>
</tr>
<tr>
<td></td>
<td>• An STC to write data to memory from DBGDTRRXint.</td>
<td></td>
<td>• An LDC to read data from memory to DBGDTRTXint.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Access to SVE, Advanced SIMD, or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in ‘EC encodings when routing exceptions to EL2’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A</td>
<td>ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN, or CPTR_ELx.TFP</td>
<td></td>
</tr>
</tbody>
</table>
architecture profile, section D1.10.4.

0b001100 Trapped MRRC access with (coproc==0b1110).

ISS encoding for an exception from an MCRR or MRRC access.

When ARMv8.5-BTI is implemented

0b001101 Branch Target Exception.

ISS encoding for an exception from Branch Target Identification instruction.

0b001110 Illegal Execution state.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault.

0b010001 SVC instruction execution in AArch32 state.

ISS encoding for an exception from HVC or SVC instruction execution.

This is reported in ESR_EL1 only when the exception is generated because the value of HCR_EL2.TGE is 1.

0b010101 SVC instruction execution in AArch64 state.

ISS encoding for an exception from HVC or SVC instruction execution.

0b011000 Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b0000000, 0b0000001 or 0b0001111.

If ARMv8.4-IDST is implemented, also exceptions generated by a read access to the feature ID space.

If ARMv8.0-CSV2 is implemented, also Cache Speculation Variant exceptions.

If ARMv8.2-EVT is implemented, also traps for EL1 and EL0 Cache controls.

This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C5.2.2, except for those exceptions reported in ESR_EL1.
using EC values 0b000000, 0b000001, or 0b000111.

**0b011001**
Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC 0b000000. This EC is defined only if SVE is implemented.

**0b011011**
Exception from an access to a TSTART instruction at EL0 when SCLTR_EL1.TME0 == 0, EL0 when SCLTR_EL2.TME0 == 0, at EL1 when SCLTR_EL1.TME == 0, at EL2 when SCLTR_EL2.TME == 0 or at EL3 when SCLTR_EL3.TME == 0.

**0b011100**
Exception from a Pointer Authentication instruction authentication failure

**0b100000**
Instruction Abort from a lower Exception level, that might be using AArch32 or AArch64. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.

**0b100001**
Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.

**0b100010**
PC alignment fault exception.
**ESR_EL1, Exception Syndrome Register (EL1)**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b100100</td>
<td>Data Abort from a lower Exception level, that might be using AArch32 or AArch64. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.</td>
</tr>
<tr>
<td>0b100101</td>
<td>Data Abort taken without a change in Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.</td>
</tr>
<tr>
<td>0b100110</td>
<td>SP alignment fault exception.</td>
</tr>
<tr>
<td>0b101000</td>
<td>Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b101100</td>
<td>Trapped floating-point exception taken from AArch64 state.</td>
</tr>
</tbody>
</table>

**ISS encoding for an exception from a Data Abort**
This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.

<table>
<thead>
<tr>
<th>EC Value</th>
<th>Description</th>
<th>ISS Encoding for an Exception</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10111</td>
<td>SError interrupt.</td>
<td><a href="#">An SError interrupt</a></td>
</tr>
<tr>
<td>0b110000</td>
<td>Breakpoint exception from a lower Exception level, that might be using AArch32 or AArch64.</td>
<td><a href="#">A Breakpoint or Vector Catch debug exception</a></td>
</tr>
<tr>
<td>0b110001</td>
<td>Breakpoint exception taken without a change in Exception level.</td>
<td><a href="#">A Breakpoint or Vector Catch debug exception</a></td>
</tr>
<tr>
<td>0b110010</td>
<td>Software Step exception from a lower Exception level, that might be using AArch32 or AArch64.</td>
<td><a href="#">A Software Step exception</a></td>
</tr>
<tr>
<td>0b110011</td>
<td>Software Step exception taken without a change in Exception level.</td>
<td><a href="#">A Software Step exception</a></td>
</tr>
<tr>
<td>0b110100</td>
<td>Watchpoint exception from a lower Exception level, that might be using AArch32 or AArch64.</td>
<td><a href="#">A Watchpoint exception</a></td>
</tr>
<tr>
<td>0b110101</td>
<td>Watchpoint exception taken without a change in Exception level.</td>
<td><a href="#">A Watchpoint exception</a></td>
</tr>
<tr>
<td>0b111000</td>
<td>BKPT instruction execution in AArch32 state.</td>
<td><a href="#">A Breakpoint instruction</a></td>
</tr>
<tr>
<td>0b11100</td>
<td>BRK instruction execution in AArch64 state. This is reported in ESR_EL3 only if a BRK instruction is executed.</td>
<td><a href="#">A Breakpoint instruction</a></td>
</tr>
</tbody>
</table>

All other EC values are reserved by Arm, and:

- Unused values in the range 0b0000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.
The effect of programming this field to a reserved value is that behavior is **CONSTRAINED UNPREDICTABLE**, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to an architecturally **UNKNOWN** value.

**IL, bit [25]**

Instruction Length for synchronous exceptions. Possible values of this bit are:

<table>
<thead>
<tr>
<th>IL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>16-bit instruction trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>32-bit instruction trapped. This value is also used when the exception is one of the following:</td>
</tr>
<tr>
<td></td>
<td>• An SError interrupt.</td>
</tr>
<tr>
<td></td>
<td>• An Instruction Abort exception.</td>
</tr>
<tr>
<td></td>
<td>• A PC alignment fault exception.</td>
</tr>
<tr>
<td></td>
<td>• An SP alignment fault exception.</td>
</tr>
<tr>
<td></td>
<td>• A Data Abort exception for which the value of the ISV bit is 0.</td>
</tr>
<tr>
<td></td>
<td>• An Illegal Execution state exception.</td>
</tr>
<tr>
<td></td>
<td>• Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning:</td>
</tr>
<tr>
<td></td>
<td>◦ 0b0: 16-bit T32 BKPT instruction.</td>
</tr>
<tr>
<td></td>
<td>◦ 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction.</td>
</tr>
<tr>
<td></td>
<td>• An exception reported using EC value 0b000000.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**ISS, bits [24:0]**

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number. For an exception taken from AArch32 state, 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not **UNPREDICTABLE**, the field takes the value 0b11111.
- If the instruction that generated the exception was **UNPREDICTABLE**, the field takes an **UNKNOWN** value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value 0b11111.

When the EC field is 0b000000, indicating an exception with an unknown reason, the ISS field is not valid, **RES0**.

**ISS encoding for exceptions with an unknown reason**

| 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | RES0 |

**Bits [24:0]**

Reserved, **RES0**.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:
• The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
  ◦ A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
  ◦ A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
  ◦ Instruction encodings that are unallocated.
  ◦ Instruction encodings for instructions that are not implemented in the implementation.
• In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
• In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
• In AArch32 state, attempted execution of a short vector floating-point instruction.
• In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
• An exception generated because of the value of one of the SCTLR_EL1, {ITD, SED, CP15BEN} control bits.
• Attempted execution of:
  ◦ An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE.
  ◦ An SMC instruction when disabled by SCR_EL3.SMD.
  ◦ An HLT instruction when disabled by EDSCR.HDE.
• Attempted execution of an MSR or MRS instruction to access SP_EL0 when the value of SPSEL.SP is 0.
• Attempted execution, in Debug state, of:
  ◦ A DCPS1 instruction when the value of HCR_EL2.TGE is 1 and EL2 is disabled or not implemented in the current Security state.
  ◦ A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
  ◦ A DCPS3 instruction when the value of EDSCR.SDD is 1, or when EL3 is not implemented.
• When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13 mon. See ‘Traps to EL3 of monitor functionality from Secure EL1 using AArch32’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
• In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
• In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
• An exception that is taken to EL2 because the value of HCR_EL2.TGE is 1 that, if the value of HCR_EL2.TGE was 0 would have been reported with an ESR_ELx.EC value of 0b000111.
  ◦ When SVE is not implemented, attempted execution of:
    ◦ An SVE instruction.
    ◦ An MSR or MRS instruction to access ZCR_EL1, ZCR_EL2, or ZCR_EL3.
• In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WFI or WFE instruction

<table>
<thead>
<tr>
<th>CV</th>
<th>COND</th>
<th>RES0</th>
<th>TI</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
</tr>
<tr>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1.
• When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.
This field resets to an architecturally unknown value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is implementation defined whether:
  - CV is set to 0 and COND is set to an unknown value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is implementation defined whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally unknown value.

**Bits [19:1]**

Reserved, RES0.

**TI, bit [0]**

Trapped instruction. Possible values of this bit are:

<table>
<thead>
<tr>
<th>TI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>WFI trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>WFE trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally unknown value.

The following fields describe configuration settings for generating this exception:

- SCTLR_EL1.{nTWE, nTWI}.
- HCR_EL2.{TWE, TWI}.
- SCR_EL3.{TWE, TWI}.

**ISS encoding for an exception from an MCR or MRC access**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | Opc2 | Opc1 | CRn | Rt | CRm | Direction |

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.
For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Opc2, bits [19:17]**

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b000.

This field resets to an architecturally UNKNOWN value.

**Opc1, bits [16:14]**

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value 0b111.

This field resets to an architecturally UNKNOWN value.

**CRn, bits [13:10]**

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

This field resets to an architecturally UNKNOWN value.

**Rt, bits [9:5]**

The Rt value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
This field resets to an architecturally **UNKNOWN** value.

**CRm, bits [4:1]**

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value **0b0000**.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRC or VMRS instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following fields describe configuration settings for generating exceptions that are reported using EC value **0b000011**:

- **CNTKCTL_EL1. {EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN},** for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **PMUSERENR_EL0. {ER, CR, SW, EN},** for accesses to Performance Monitor registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **AMUSERENR_EL0. EN,** for accesses to Activity Monitors registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **HCR_EL2. {TRVM, TVM},** for accesses to virtual memory control registers from EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **HCR_EL2. TCL, TCTL, TSW, TPC, TPU, TACR, TIDCP, CPTR_EL2.TAM, HSTR_EL2. T<n>,** for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- **HCR_EL2. TIDCP, for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL2.TCPAC, for accesses to CPACR_EL1 or CPACR using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- **HSTR_EL2. T<n>,** for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

The following fields describe configuration settings for generating exceptions that are reported using EC value **0b000101**:

- **ESR_EL1, Exception Syndrome Register (EL1)**
• **CPACR_EL1.TTA** for accesses to trace registers, MCR or MRC access (coproc == 0b11110) trapped to EL1 or EL2.

• **MDSCR_EL1.TDC,** for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.

• If ARMv8.6-FGT is implemented, **MDCR_EL2.TDCC** and **MDCR_EL3.TDCC,** for accesses to the DCC registers at EL0 and EL1, trapped to EL2.

• **HCR_EL2.TID0,** for accesses to the **IIDR** register in the ID group 0 at EL0 and EL1 using AArch32, MRC access (coproc == 0b11110) trapped to EL2.

• **CPTR_EL2.TTA,** for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b11110) trapped to EL2.

• **MDCR_EL2.TDRA,** for accesses to Debug ROM registers **DBGDRAR** and AArch-DBGDSAR using AArch32, MCR or MRC access (coproc == 0b11110) trapped to EL2.

• **MDCR_EL2.TDOSA,** for accesses to powerdown debug registers, using AArch32 state, MCR or MRC access (coproc == 0b11110) trapped to EL2.

• **CPTR_EL2.TDA,** for accesses to other debug registers, using AArch32 state, MCR or MRC access (coproc == 0b11110) trapped to EL2.

• **MDCR_EL3.TDOSA,** for accesses to powerdown debug registers using AArch32, MCR or MRC access (coproc == 0b11110) trapped to EL3.

• **MDCR_EL3.TDA,** for accesses to other debug registers, using AArch32, MCR or MRC access (coproc == 0b11110) trapped to EL3.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001000:

• **HCR_EL2.TID0,** for accesses to the **FPSID** register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.

• **HCR_EL2.TID3,** for accesses to registers in ID group 3 including **MVFR0, MVFR1** and **MVFR2,** VMRS access trapped to EL2.

### ISS encoding for an exception from an MCRR or MRRC access

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>COND</td>
<td>Opc1</td>
<td>RES0</td>
<td>Rt2</td>
<td>Rt</td>
<td>CRm</td>
<td>Direction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1.

• When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to \(0b1110\), the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is implementation defined whether:

- CV is set to 0 and COND is set to an unknown value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is implementation defined whether the COND field is set to \(0b1110\), or to the value of any condition that applied to the instruction.

This field resets to an architecturally unknown value.

**Opc1, bits [19:16]**

The Opc1 value from the issued instruction.

This field resets to an architecturally unknown value.

**Bit [15]**

Reserved, RES0.

**Rt2, bits [14:10]**

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally unknown value.

**Rt, bits [9:5]**

The Rt value from the issued instruction, the first general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally unknown value.

**CRm, bits [4:1]**

The CRm value from the issued instruction.

This field resets to an architecturally unknown value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCRR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRRC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally unknown value.

The following fields describe configuration settings for generating exceptions that are reported using EC value \(0b000100\):

- **CNTKCTL_EL1** (EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN), for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **PMUSERENR_EL0** (CR, EN), for accesses to Performance Monitor registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
The following sections describe configuration settings for generating exceptions that are reported using EC value 0b001100:

- **CPACR_EL1.TTA** for accesses to trace registers using MCR or MRC instructions, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- **MDSCR_EL1.TDCC**, for accesses to the Debug Communications Channel (DCC) registers DBGDSAR and DBGDRAR at EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **CPTR_EL2.TTA**, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **MDCR_EL2.TDRA**, for accesses to Debug ROM registers DBGDRAR and AArch-DBGDSAR using AArch32, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL3.TTA**, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- **MDCR_EL3.TDOA**, for traps to powerdown debug registers using AArch32, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- **MDCR_EL3.TDA**, for accesses to other debug registers, using AArch32, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.

### ISS encoding for an exception from an LDC or STC instruction

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>COND</td>
<td>imm8</td>
<td>RES0</td>
<td>Rn</td>
<td>Offset</td>
<td>AM</td>
<td>Direction</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.
For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**imm8, bits [19:12]**

The immediate value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

**Bits [11:10]**

Reserved, RES0.

**Rn, bits [9:5]**

The Rn value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**Offset, bit [4]**

Indicates whether the offset is added or subtracted:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Subtract offset.</td>
</tr>
<tr>
<td>0b1</td>
<td>Add offset.</td>
</tr>
</tbody>
</table>

This bit corresponds to the U bit in the instruction encoding.

This field resets to an architecturally UNKNOWN value.

**AM, bits [3:1]**

Addressing mode. The permitted values of this field are:
<table>
<thead>
<tr>
<th>AM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Immediate unindexed.</td>
</tr>
<tr>
<td>0b001</td>
<td>Immediate post-indexed.</td>
</tr>
<tr>
<td>0b010</td>
<td>Immediate offset.</td>
</tr>
<tr>
<td>0b011</td>
<td>Immediate pre-indexed.</td>
</tr>
<tr>
<td>0b100</td>
<td>For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.</td>
</tr>
<tr>
<td>0b110</td>
<td>For a trapped STC instruction, this encoding is reserved.</td>
</tr>
</tbody>
</table>

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is **CONSTRAINED UNPREDICTABLE**, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits \{P, W\} in the instruction encoding.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to memory. STC instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from memory. LDC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000110:

- **MDSCR_EL1.TDCC**, for accesses using AArch32 state, LDC access to **DBGDTRXInt** or STC access to **DBGDTRXInt** trapped to EL1 or EL2.
- **MDCR_EL2.TDA**, for accesses using AArch32 state, LDC access to **DBGDTRXInt** or STC access to **DBGDTRXInt** MCR or MRC access trapped to EL2.
- **MDCR_EL3.TDA**, for accesses using AArch32 state, LDC access to **DBGDTRXInt** or STC access to **DBGDTRXInt** MCR or MRC access trapped to EL3.
- If ARMv8.6-FGT is implemented, **MDCR_EL2.TDCC** and **MDCR_EL3.TDCC**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.

**ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN or CPTR_ELx.TFP**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | RE50 |

The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.

For an implementation that does not include either SVE or support for floating-point and Advanced SIMD, the exception is reported using the EC value 0b000000.

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.
For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Bits [19:0]**

Reserved, RES0.

The following sections describe the configuration settings for the traps that are reported using EC value 0b000111:

- CPACR_EL1.FPEN, for accesses to SIMD and floating-point registers trapped to EL1.
- CPTR_EL2.TFP, for accesses to SIMD and floating-point registers trapped to EL2.
- CPTR_EL2.TFP, for accesses to SIMD and floating-point registers trapped to EL3.

**ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

**Bits [24:0]**

When SVE is implemented:

Reserved, RES0.

Otherwise:

Reserved, RES0.
The accesses covered by this trap include:

- Execution of SVE instructions.
- Accesses to the SVE system registers, ZCR_ELx and ID_AA64ZFR0_EL1.

For an implementation that does not include SVE, the exception is reported using the EC value 0b000000.

**ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [24:0]**

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions see 'The Illegal Execution state exception' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile and 'PC alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

‘Stack pointer alignment checking’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the configuration settings for generating SP alignment fault exceptions.

**ISS encoding for an exception from HVC or SVC instruction execution**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | imm16 |

**Bits [24:16]**

Reserved, RES0.

**imm16, bits [15:0]**

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.


For A64 instructions, see ‘SVC’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile and ‘HVC’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If ARMv8.6-FGT is implemented, HFGITR_EL2, {SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.
ISS encoding for an exception from SMC instruction execution in AArch32 state

24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
CV  COND  CCKNOWNPASS  RES0

For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because HCR_EL2.TSC is 1, the ISS encoding is as shown in the diagram.

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.

**CCKNOWNPASS, bit [19]**

Indicates whether the instruction might have failed its condition code check.
### CKNOWNPASS

<table>
<thead>
<tr>
<th>CKNOWNPASS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The instruction was unconditional, or was conditional and passed its condition code check.</td>
</tr>
<tr>
<td>0b1</td>
<td>The instruction was conditional, and might have failed its condition code check.</td>
</tr>
</tbody>
</table>

#### Note

In an implementation in which an SMC instruction that fails its code check is not trapped, this field can always return the value 0.

This field resets to an architecturally **UNKNOWN** value.

### Bits [18:0]

Reserved, **RES0**.

HCR_EL2.TSC describes the configuration settings for trapping SMC instructions to EL2.

See ‘System calls’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the case where these exceptions are trapped to EL3.

### ISS encoding for an exception from SMC instruction execution in AArch64 state

<table>
<thead>
<tr>
<th>Bit positions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24–22</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>15–0</td>
<td><em>imm16</em></td>
</tr>
</tbody>
</table>

#### Bits [24:16]

Reserved, **RES0**.

**imm16**, bits [15:0]

The value of the immediate field from the issued SMC instruction.

This field resets to an architecturally **UNKNOWN** value.

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

HCR_EL2.TSC describes the configuration settings for trapping SMC from EL1 modes.

‘System calls’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the case where these exceptions are trapped to EL3.

### ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

<table>
<thead>
<tr>
<th>Bit positions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24–22</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>21–10</td>
<td><strong>Op0</strong></td>
</tr>
<tr>
<td>19–8</td>
<td><strong>Op2</strong></td>
</tr>
<tr>
<td>17–6</td>
<td><strong>Op1</strong></td>
</tr>
<tr>
<td>15–4</td>
<td><strong>CRn</strong></td>
</tr>
<tr>
<td>3–0</td>
<td><strong>Rt</strong></td>
</tr>
<tr>
<td>23</td>
<td><strong>CRm</strong></td>
</tr>
<tr>
<td>22</td>
<td><strong>Direction</strong></td>
</tr>
</tbody>
</table>

#### Bits [24:22]

Reserved, **RES0**.
Op0, bits [21:20]

The Op0 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

This field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write access, including MSR instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read access, including MRS instructions.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

For exceptions caused by System instructions, see 'System' subsection of 'Branches, exception generating and System instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for the encoding values returned by an instruction.

The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- **SCTLR_EL1.UCI**, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.UCT**, for accesses to **CTR_EL0** using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.DZE**, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.UMA**, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **CPACR_EL1.TTA**, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
• **MDSCR_EL1.TDCC**, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
• If ARMv8.6-FGT is implemented, **MDSCR_EL2.TDCC** and **MDSCR_EL3.TDCC**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.
• **CNTKCTL_EL1.EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN** accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
• **PMUSERENR_EL0** (ER, CR, SW, EN), for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
• **AMUSERENR_EL0.** EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
• **HCR_EL2.TRDV**, for accesses to the Auxiliary Control Register, **ACTLR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
• **HCR_EL2.TDCP**, for accesses to lock-down, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
• **HCR_EL2.TID1, TID2, TID3**, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
• **CPTR_EL2.TCPAC**, for accesses to **CPACR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
• **CPTR_EL2.TTA**, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.TTRF**, for accesses to the trace filter register, **TRFCR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.TDRA**, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.TDOSA**, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
• **CNTKCTL_EL2.EL1PCEN, EL1PCTEN**, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.TDA**, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.** {TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.TAM**, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
• **HCR_EL2.APK**, for accesses to Pointer authentication key registers. using AArch64 state, MSR or MRS access trapped to EL2.
• **HCR_EL2.NV, NV1**, for Nested virtualization register access, using AArch64 state, MSR or MRS access trapped to EL2.
• **HCR_EL2.AT**, for execution of AT S1E* instructions, using AArch64 state, MSR or MRS access, trapped to EL2.
• **HCR_EL2.TERR, FIEN**, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
• **SCP_EL3.** APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **SCP_EL3.ST**, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **SCP_EL3.TERR, FIEN**, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **CPTR_EL3.TCPAC**, for accesses to **CPTR_EL2** and **CPACR_EL1** using AArch64 state, MSR or MRS access trapped to EL3.
• **CPTR_EL3.TTA**, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **MDCR_EL3.TTRF**, for accesses to the filter trace control registers, **TRFCR_EL1** and **TRFCR_EL2**, using AArch64 state, MSR or MRS access trapped to EL3.
• **MDCR_EL3.TDA**, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **MDCR_EL3.TDOSA**, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **MDCR_EL3.TPM**, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
• **CPTR_EL3.TAM**, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.
• If ARMv8.2-EVT is implemented, \texttt{HCR\_EL2\{TTLBOS, TTLBIS, TICAB, TOCU, TID4\}} and \texttt{HCR\_EL2\{TTLBIS, TICAB, TOCU, TID4\}} control traps for EL1 and EL0 Cache controls that use this EC value.

• If ARMv8.6-FGT is implemented:
  - \texttt{SCR\_EL3\_FGTEn}, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - \texttt{HFGRTR\_EL2} for reads and \texttt{HFGWTR\_EL2} for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
  - \texttt{HFGITR\_EL2} for execution of system instructions, MSR or MRS access trapped to EL2.
  - \texttt{HDFGRTR\_EL2} for reads and \texttt{HDFGWTR\_EL2} for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
  - \texttt{HAFGRTR\_EL2} for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.

**ISS encoding for an IMPLEMENTATION DEFINED exception to EL3**

\begin{verbatim}
  24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
  IMPLEMENTATION DEFINED
\end{verbatim}

**IMPLEMENTATION DEFINED, bits [24:0]**

- This field resets to an architecturally UNKNOWN value.

**ISS encoding for an exception from an Instruction Abort**

\begin{verbatim}
  24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
  RES0 SET FnV EA RES0 S1PTW RES0 IFSC
\end{verbatim}

**Bits [24:13]**

- Reserved, RES0.

**SET, bits [12:11]**

Synchronous Error Type. When the RAS Extension is implemented and IFSC is 0b010000, describes the state of the PE after taking the Instruction Abort exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>SET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b11</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Note**

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the IFSC field is not 0b010000.

This field resets to an architecturally UNKNOWN value.
**FnV, bit [10]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>FAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid if the IFSC code is 0b010000. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

**EA, bit [9]**

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.

**Bit [8]**

Reserved, RES0.

**S1PTW, bit [7]**

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>

For any abort other than a stage 2 fault this bit is RES0.

This field resets to an architecturally UNKNOWN value.

**Bit [6]**

Reserved, RES0.

**IFSC, bits [5:0]**

Instruction Fault Status Code. Possible values of this field are:
### IFSC Encoding

<table>
<thead>
<tr>
<th>IFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault, level 0 of translation or translation table base register</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access flag fault, level 1</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 2</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 3</td>
</tr>
<tr>
<td>0b001011</td>
<td>Access flag fault, level 3</td>
</tr>
<tr>
<td>0b001100</td>
<td>Permission fault, level 1</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 2</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 2</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 0</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk</td>
</tr>
<tr>
<td>0b011100</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 0</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM. Otherwise reserved.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

---

**Note**

Armv8.2 requires the implementation of the RAS Extension.

---

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

---

**Note**

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

---

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally **UNKNOWN** value.

---

**ISS encoding for an exception from a Data Abort**
ISV, bit [24]

Instruction syndrome valid. Indicates whether the syndrome information in ISS[23:14] is valid.

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No valid instruction syndrome. ISS[23:14] are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>ISS[23:14] hold a valid instruction syndrome.</td>
</tr>
</tbody>
</table>

This bit is 0 for all faults reported in ESR_EL2 except the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback.
- AArch32 instructions where the instruction:
  ○ Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
  ○ Is not performing register writeback.
  ○ Is not using R15 as a source or destination register.

For these cases, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

ISV is 0 for all faults reported in ESR_EL1 or ESR_EL3.

When the RAS Extension is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When the RAS Extension is not implemented, the value of ISV on a synchronous External abort on a stage 2 translation table walk is IMPLEMENTATIONDEFINED.

This field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

<table>
<thead>
<tr>
<th>SAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Byte</td>
</tr>
<tr>
<td>0b01</td>
<td>Halfword</td>
</tr>
<tr>
<td>0b10</td>
<td>Word</td>
</tr>
<tr>
<td>0b11</td>
<td>Doubleword</td>
</tr>
</tbody>
</table>

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

SSE, bit [21]

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sign-extension not required.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data item must be sign-extended.</td>
</tr>
</tbody>
</table>

For all other operations this bit is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.
This field resets to an architecturally **UNKNOWN** value.

**SRT, bits [20:16]**

Syndrome Register transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction. If the exception was taken from an Exception level that is using AArch32 then this is the AArch64 view of the register. See ‘Mapping of the general-purpose registers between the Execution states’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**SF, bit [15]**

Width of the register accessed by the instruction is Sixty-Four. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction loads/stores a 32-bit wide register.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction loads/stores a 64-bit wide register.</td>
</tr>
</tbody>
</table>

**Note**

This field specifies the register width identified by the instruction, not the Execution state.

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**AR, bit [14]**

Acquire/Release. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction did not have acquire/release semantics.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction did have acquire/release semantics.</td>
</tr>
</tbody>
</table>

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**VNCR, bit [13]**

When ARMv8.4-NV is implemented:

Indicates that the fault came from use of **VNCR_EL2** register by EL1 code.

<table>
<thead>
<tr>
<th>VNCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault was not generated by the use of <strong>VNCR_EL2</strong>, by an MRS or MSR instruction executed at EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault was generated by the use of <strong>VNCR_EL2</strong>, by an MRS or MSR instruction executed at EL1.</td>
</tr>
</tbody>
</table>

This field is 0 in ESR_EL1.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**SET, bits [12:11]**

Synchronous Error Type. When the RAS Extension is implemented and DFSC is 0b010000, describes the state of the PE after taking the Data Abort exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>SET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b11</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Note**

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010000.

This field resets to an architecturally UNKNOWN value.

**FnV, bit [10]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>FAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

**EA, bit [9]**

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.

**CM, bit [8]**

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:
CM

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 The Data Abort was not generated by the execution of one of the</td>
</tr>
<tr>
<td>System instructions identified in the description of value 1.</td>
</tr>
<tr>
<td>0b1 The Data Abort was generated by either the execution of a</td>
</tr>
<tr>
<td>cache maintenance instruction or by a synchronous fault on the</td>
</tr>
<tr>
<td>execution of an address translation instruction. The DC ZVA instruction</td>
</tr>
<tr>
<td>is not classified as a cache maintenance instruction, and therefore</td>
</tr>
<tr>
<td>its execution cannot cause this field to be set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>

For any abort other than a stage 2 fault this bit is RES0.

This field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort caused by an instruction reading from a memory location.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by an instruction writing to a memory location.</td>
</tr>
</tbody>
</table>

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

This field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code. Possible values of this field are:
### DFSC

<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault, level 0 of translation or translation table base register.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b001011</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001100</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b010001</td>
<td>Synchronous Tag Check fail.</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 0.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011001</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 0.</td>
</tr>
<tr>
<td>0b011010</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011100</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100000</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM. Otherwise reserved.</td>
</tr>
<tr>
<td>0b111100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown).</td>
</tr>
<tr>
<td>0b111101</td>
<td>IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).</td>
</tr>
<tr>
<td>0b111110</td>
<td>Section Domain Fault, used only for faults reported in the PAR_EL1.</td>
</tr>
<tr>
<td>0b111111</td>
<td>Page Domain Fault, used only for faults reported in the PAR_EL1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally **UNKNOWN** value.
ISS encoding for an exception from a trapped floating-point exception

| 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | TFV | RES0 | VECITR | IDF | RES0 | IXF | UFF | OFF | DZF | IOF |

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions. The possible values of this bit are:

<table>
<thead>
<tr>
<th>TFV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN.</td>
</tr>
<tr>
<td>0b1</td>
<td>One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information see 'Floating-point exception traps' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from a vector instruction.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from a vector instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

This field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IDF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input denormal floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input denormal floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
**Bits [6:5]**

Reserved, RES0.

**IXF, bit [4]**

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is **UNKNOWN**. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IXF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Inexact floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inexact floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**UFF, bit [3]**

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is **UNKNOWN**. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>UFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**OFF, bit [2]**

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is **UNKNOWN**. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>OFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**DZF, bit [1]**

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is **UNKNOWN**. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>DZF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Divide by Zero floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Divide by Zero floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**IOF, bit [0]**

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is **UNKNOWN**. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IOF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Invalid Operation floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invalid Operation floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.
In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the \texttt{FPSCR.\{IDE, IXE, UFE, OFE, DZE, IOE\}} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the \texttt{FPSCR.\{IDE, IXE, UFE, OFE, DZE, IOE\}} bits enable each of the floating-point exception traps.

**ISS encoding for an SError interrupt**

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>IDS</td>
<td>RES0</td>
<td>IESB</td>
<td>AET</td>
<td>EA</td>
<td>RES0</td>
<td>DFSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IDS, bit [24]**

**IMPLEMENTATION DEFINED** syndrome. Possible values of this bit are:

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bits[23:0] of the ISS field holds the fields described in this encoding.</td>
</tr>
</tbody>
</table>

**Note**

If the RAS Extension is not implemented, this means that bits[23:0] of the ISS field are \texttt{RES0}.

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Bits[23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt.</td>
</tr>
</tbody>
</table>

**Note**

This field was previously called ISV.

This field resets to an architecturally \texttt{UNKNOWN} value.

**Bits [23:14]**

Reserved, \texttt{RES0}.

**IESB, bit [13]**

**When ARMv8.2-IESB is implemented:**

Implicit error synchronization event.

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately.</td>
</tr>
<tr>
<td>0b1</td>
<td>The SError interrupt was synchronized by the implicit error synchronization event and taken immediately.</td>
</tr>
</tbody>
</table>

This field is \texttt{RES0} if the value returned in the DFSC field is not \texttt{0b010001}.

**Note**

Armv8.2 requires the implementation of the RAS Extension and ARMv8.2-IESB.

This field resets to an architecturally \texttt{UNKNOWN} value.
Otherwise:

Reserved, RES0.

**AET, bits [12:10]**

Asynchronous Error Type.

When the RAS Extension is implemented and DFSC is 0b010001, describes the state of the PE after taking the SError interrupt exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>AET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Uncontainable error (UC)</td>
</tr>
<tr>
<td>0b001</td>
<td>Unrecoverable error (UEU)</td>
</tr>
<tr>
<td>0b010</td>
<td>Restartable error (UEO)</td>
</tr>
<tr>
<td>0b011</td>
<td>Recoverable error (UER)</td>
</tr>
<tr>
<td>0b110</td>
<td>Corrected error (CE)</td>
</tr>
</tbody>
</table>

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall state of the PE is reported. For example, if both a Recoverable and Unrecoverable error occurred, the state is Unrecoverable.

**Note**

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally **UNKNOWN** value.

**EA, bit [9]**

External abort type. When the RAS Extension is implemented, this bit can provide an **IMPLEMENTATION DEFINED** classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally **UNKNOWN** value.

**Bits [8:6]**

Reserved, RES0.
**DFSC, bits [5:0]**

Data Fault Status Code. When the RAS Extension is implemented, possible values of this field are:

<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Uncategorized.</td>
</tr>
<tr>
<td>0b010001</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If the RAS Extension is not implemented, this field is RES0.

---

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

---

**ISS encoding for an exception from a Breakpoint or Vector Catch debug exception**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

** Bits [24:6]**

Reserved, RES0.

---

**IFSC, bits [5:0]**

Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

---

**ISS encoding for an exception from a Software Step exception**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| ISV |    | RES0 |    |    | EX  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**ISV, bit [24]**

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EX bit is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>EX bit is valid.</td>
</tr>
</tbody>
</table>

See the EX bit description for more information.

This field resets to an architecturally UNKNOWN value.
**Bits [23:7]**

Reserved, RES0.

**EX, bit [6]**

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

<table>
<thead>
<tr>
<th>EX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An instruction other than a Load-Exclusive instruction was stepped.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Load-Exclusive instruction was stepped.</td>
</tr>
</tbody>
</table>

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

This field resets to an architecturally **UNKNOWN** value.

**IFSC, bits [5:0]**

Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally **UNKNOWN** value.

For more information about generating these exceptions, see 'Software Step exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**ISS encoding for an exception from a Watchpoint exception**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>VNCR</td>
<td>RES0</td>
<td>CM</td>
<td>RES0</td>
<td>WnR</td>
<td>DFSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [24:14]**

Reserved, RES0.

**VNCR, bit [13]**

When ARMv8.4-NV is implemented:

Indicates that the watchpoint came from use of **VNCR_EL2** register by EL1 code.

<table>
<thead>
<tr>
<th>VNCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The watchpoint was not generated by the use of <strong>VNCR_EL2</strong> by EL1 code.</td>
</tr>
<tr>
<td>0b1</td>
<td>The watchpoint was generated by the use of <strong>VNCR_EL2</strong> by EL1 code.</td>
</tr>
</tbody>
</table>

This field is 0 in ESR_EL1.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Bits [12:9]**

Reserved, RES0.
Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction. The DC_ZVA instruction is not classified as a cache maintenance instruction, and therefore its execution cannot cause this field to be set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bit [7]**

Reserved, RES0.

**WnR, bit [6]**

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint exception caused by an instruction reading from a memory location.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint exception caused by an instruction writing to a memory location.</td>
</tr>
</tbody>
</table>

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is **UNPREDICTABLE** which watchpoint generates the Watchpoint exception.

This field resets to an architecturally **UNKNOWN** value.

**DFSC, bits [5:0]**

Data Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally **UNKNOWN** value.

For more information about generating these exceptions, see ‘Watchpoint exceptions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**ISS encoding for an exception from execution of a Breakpoint instruction**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|--|--|--|--|--|--|--|
| RES0 | Comment |

**Bits [24:16]**

Reserved, RES0.
**Comment, bits [15:0]**

Set to the instruction comment field value, zero extended as necessary. For the AArch32 BKPT instructions, the comment field is described as the immediate field.

This field resets to an architecturally **UNKNOWN** value.

For more information about generating these exceptions, see 'Breakpoint instruction exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**ISS encoding for an exception from ERET, ERETTA or ERETAB instruction**

```
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
<table>
<thead>
<tr>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Res0</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ERET</td>
<td>ERETA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

This EC value applies when ARMv8.6-FGT is implemented, or when **HCR_EL2**.NV is 1.

**Bits [24:2]**

Reserved, **RES0**.

**ERET, bit [1]**

Indicates whether an ERET or ERETTA* instruction was trapped to EL2. Possible values are:

<table>
<thead>
<tr>
<th>ERET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERET instruction trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERETTA or ERETAB instruction trapped to EL2.</td>
</tr>
</tbody>
</table>

If this bit is 0, the ERETTA field is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

**ERE T A, bit [0]**

Indicates whether an ERETTA or ERETAB instruction was trapped to EL2. Possible values are:

<table>
<thead>
<tr>
<th>ERETTA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERETTA instruction trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERETAB instruction trapped to EL2.</td>
</tr>
</tbody>
</table>

When the ERET field is 0, this bit is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

For more information about generating these exceptions, see **HCR_EL2**.NV.

If ARMv8.6-FGT is implemented, **HFGITR_EL2**.ERET controls fine-grained trap exceptions from ERET, ERETTA and ERETAB execution.

**ISS encoding for an exception from a TSTART instruction**

```
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
<table>
<thead>
<tr>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
<th>bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Rd</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Bits [24:10]**

Reserved, **RES0**.

**Rd, bits [9:5]**

The Rd value from the issued instruction, the general purpose register used for the destination.
Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BTYPE |

Bits [24:2]

Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see 'The AArch64 application level programmers model' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section B1. 

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |

Bits [24:0]

Reserved, RES0.

For more information about generating these exceptions, see:

- **HCR_EL2.API**, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- **SCR_EL3.API**, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for an exception from a Pointer Authentication instruction authentication failure

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Bit [1] Bit [0] |

Bits [24:2]

Reserved, RES0.

Bit [1], bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

<table>
<thead>
<tr>
<th>Bit [1]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction Key.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data Key.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
**Bit [0], bit [0]**

This field indicates whether the exception is as a result of an A key or a B key.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A key.</td>
</tr>
<tr>
<td>0b1</td>
<td>B key.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following instructions generate an exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTIASP, AUTIAZ, AUTIA1716.
- AUTIBSP, AUTIBZ, AUTIB1716.
- AUTIA, AUTDA, AUTIB, AUTDB.
- AUTIZA, AUTIZB, AUTDZA, AUTDZB.

It is **IMPLEMENTATION DEFINED** whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a translation fault when the address is accessed:

- RETAA, RETAB.
- BRAA, BRAB, BLRAA, BLRAB.
- BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERATA, ERATB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

### Accessing the ESR_EL1

**ESR_EL1, Exception Syndrome Register (EL1)**

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL3 using the mnemonic ESR_EL1 or ESR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, ESR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```assembler
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x138];
    else
        return ESR_EL1;
    end if;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ESR_EL2;
    else
        return ESR_EL1;
    end if;
elsif PSTATE.EL == EL3 then
    return ESR_EL1;
end if;
```

**MSR ESR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x138] = X[t];
    else
        ESR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            ESR_EL2 = X[t];
        else
            ESR_EL1 = X[t];
        elsif PSTATE.EL == EL3 then
            ESR_EL1 = X[t];
        MSR ESR_EL12, <Xt>
        op0  op1  CRn  CRm  op2
        0b11  0b101  0b0101  0b0010  0b000
    if PSTATE.EL == EL0 then
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x138];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return ESR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return ESR_EL1;
    else
        UNDEFINED;
MSR ESR_EL12, <Xt>
        op0  op1  CRn  CRm  op2
        0b11  0b101  0b0101  0b0010  0b000
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x138] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    ESR_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    ESR_EL1 = X[t];
  else
    UNDEFINED;
MRS <Xt>, ESR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return ESR_EL1;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  ESR_EL2 = X[t];

MSR ESR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    ESR_EL1 = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  ESR_EL2 = X[t];
ESR_EL2, Exception Syndrome Register (EL2)

The ESR_EL2 characteristics are:

**Purpose**

Holds syndrome information for an exception taken to EL2.

**Configuration**

AArch64 System register ESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ESR_EL2 is a 64-bit register.

**Field descriptions**

The ESR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>EC</td>
<td>IL</td>
<td>RES0</td>
<td>ISS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

ESR_EL2 is made UNKNOWN as a result of an exception return from EL2.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL2, the value of ESR_EL2 is UNKNOWN. The value written to ESR_EL2 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

**Bits [63:32]**

Reserved, RES0.

**EC, bits [31:26]**

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:
<table>
<thead>
<tr>
<th>EC</th>
<th>Meaning</th>
<th>ISS Encoding for exceptions with an unknown reason</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Unknown reason.</td>
<td>ISS encoding for an exception from a WFI or WFE instruction</td>
<td>Applies when unknown reason.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Trapped WFI or WFE instruction execution. Conditional WFE and WFI instructions that fail their condition code check do not cause an exception.</td>
<td>ISS encoding for an exception from an MCRR or MRRC access</td>
<td>Applies when Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for an exception from an MCR or MRC access</td>
<td>Applies when Trapped MCRR or MRRC access with (coproc==0b1111).</td>
</tr>
<tr>
<td>0b000011</td>
<td>Trapped MCR or MRC access with (coproc==0b1110).</td>
<td>ISS encoding for an exception from an MCRR or MRC access</td>
<td>Applies when Trapped MCR or MRC access with (coproc==0b1110).</td>
</tr>
<tr>
<td>0b000100</td>
<td>Trapped LDC or STC access. The only architected uses of these instruction are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.</td>
<td>ISS encoding for an exception from an LDC or STC instruction</td>
<td>Applies when Trapped LDC or STC access.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Access to SVE, Advanced SIMD, or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in ‘EC encodings when routing exceptions to EL2’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A.</td>
<td>ISS encoding for an access to SVE, Advanced SIMD or floating-point functionality resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control.</td>
<td>Applies when Access to SVE, Advanced SIMD, or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in ‘EC encodings when routing exceptions to EL2’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A.</td>
</tr>
</tbody>
</table>
architecture profile, section D1.10.4.

Trapped VMRS access, from ID group trap, that is not reported using EC 0b000111.

ISS encoding for an exception from an MCR or MRC access

Trapped use of a Pointer authentication instruction because HCR_EL2.API == 0 || SCR_EL3.API == 0.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0.

Trapped MRRC access with (coproc==0b1110).

ISS encoding for an exception from an MCRR or MRRC access

when ARMv8.3-PAuth is implemented

Branch Target Exception.

ISS encoding for an exception from Branch Target Identification instruction

when ARMv8.5-BTI is implemented

Illegal Execution state.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault

when ARMv8.3-PAuth is implemented

SVC instruction execution in AArch32 state.

ISS encoding for an exception from HVC or SVC instruction execution

This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TGE is 1.

HVC instruction execution in AArch32 state, when HVC is not disabled.

ISS encoding for an exception from HVC or SVC instruction execution

SMC instruction execution in AArch32 state, when SMC is not disabled.

ISS encoding for an exception from SMC instruction execution in AArch32 state

This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1.

SVC instruction execution in AArch64 state.

ISS encoding for an exception from HVC or SVC instruction execution

HVC instruction execution in AArch64 state, when HVC is not disabled.

ISS encoding for an exception from HVC or SVC instruction execution

SMC instruction execution in AArch64 state, when SMC is not disabled.

ISS encoding for an exception from SMC instruction execution in AArch64 state

This is reported in ESR_EL2 only when
the exception is generated because the value of HCR_EL2.TSC is 1.

0b011000
Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b000000, 0b000001 or 0b000111.
If ARMv8.4-IDST is implemented, also exceptions generated by a read access to the feature ID space.
If ARMv8.0-CSV2 is implemented, also Cache Speculation Variant exceptions.
If ARMv8.2-EVT is implemented, also traps for EL1 and EL0 Cache controls.
This includes all instructions that cause exceptions that are part of the encoding space defined in ‘System instruction class encoding overview’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C5.2.2, except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111.

0b011001
Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC 0b000000.
This EC is defined only if SVE is implemented.

0b011010
Trapped ERET, ERETAA, or ERETAB instruction execution.

0b011011
Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when
SCTLR_EL2.TME0 == 0, at EL1 when
SCTLR_EL1.TME == 0, at EL2 when
SCTLR_EL2.TME == 0 or at EL3 when
SCTLR_EL3.TME == 0.

0b011100 Exception from a Pointer Authentication instruction authentication failure
ISS encoding for an exception from a Pointer Authentication instruction authentication failure
When ARMv8.3-FPAC is implemented

0b100000 Instruction Abort from a lower Exception level, that might be using AArch32 or AArch64. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.
ISS encoding for an exception from an Instruction Abort

0b100001 Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.
ISS encoding for an exception from an Instruction Abort

0b100010 PC alignment fault exception.
ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault

0b100100 Data Abort from a lower Exception level, excluding Data Aborts taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. These Data Aborts might be generated from Exception levels using AArch32 or AArch64.
ISS encoding for an exception from a Data Abort
<table>
<thead>
<tr>
<th>EC Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b100101</td>
<td>Data Abort without a change in Exception level, or Data Aborts taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug related exceptions.</td>
</tr>
<tr>
<td>0b100110</td>
<td>SP alignment fault exception.</td>
</tr>
<tr>
<td>0b101000</td>
<td>Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b101100</td>
<td>Trapped floating-point exception taken from AArch64 state.</td>
</tr>
</tbody>
</table>
This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED.

0b101111 SError interrupt. ISS encoding for an SError interrupt

0b110000 Breakpoint exception from a lower Exception level, that might be using AArch32 or AArch64. ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

0b110001 Breakpoint exception taken without a change in Exception level. ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

0b110010 Software Step exception from a lower Exception level, that might be using AArch32 or AArch64. ISS encoding for an exception from a Software Step exception

0b110011 Software Step exception taken without a change in Exception level. ISS encoding for an exception from a Software Step exception

0b110100 Watchpoint from a lower Exception level, excluding Watchpoint Exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. ISS encoding for an exception from a Watchpoint exception

0b110101 Watchpoint exceptions without a change in Exception level, or Watchpoint exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. ISS encoding for an exception from a Watchpoint exception
0b111000 BKPT instruction execution in AArch32 state. ISS encoding for an exception from execution of a Breakpoint instruction

0b111010 Vector Catch exception from AArch32 state. The only case where a Vector Catch exception is taken to an Exception level that is using AArch64 is when the exception is routed to EL2 and EL2 is using AArch64. ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

0b111100 BRK instruction execution in AArch64 state. ISS encoding for an exception from execution of a Breakpoint instruction

All other EC values are reserved by Arm, and:

• Unused values in the range 0b00000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
• Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

<table>
<thead>
<tr>
<th>IL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>16-bit instruction trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>32-bit instruction trapped. This value is also used when the exception is one of the following:</td>
</tr>
<tr>
<td></td>
<td>• An SError interrupt.</td>
</tr>
<tr>
<td></td>
<td>• An Instruction Abort exception.</td>
</tr>
<tr>
<td></td>
<td>• A PC alignment fault exception.</td>
</tr>
<tr>
<td></td>
<td>• An SP alignment fault exception.</td>
</tr>
<tr>
<td></td>
<td>• A Data Abort exception for which the value of the ISV bit is 0.</td>
</tr>
<tr>
<td></td>
<td>• An Illegal Execution state exception.</td>
</tr>
<tr>
<td></td>
<td>• Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning:</td>
</tr>
<tr>
<td></td>
<td>• 0b0: 16-bit T32 BKPT instruction.</td>
</tr>
<tr>
<td></td>
<td>• 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction.</td>
</tr>
<tr>
<td></td>
<td>• An exception reported using EC value 0b000000.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.
Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number. For an exception taken from AArch32 state, ‘Mapping of the general-purpose registers between the Execution states’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If the AArch32 register descriptor is \(0b1111\), then:

- If the instruction that generated the exception was not \textit{UNPREDICTABLE}, the field takes the value \(0b11111\).
- If the instruction that generated the exception was \textit{UNPREDICTABLE}, the field takes an \textit{UNKNOWN} value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value \(0b11111\).

When the EC field is \(0b000000\), indicating an exception with an unknown reason, the ISS field is not valid, \texttt{RES0}.

**ISS encoding for exceptions with an unknown reason**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

\textbf{Bits [24:0]}

Reserved, \texttt{RES0}.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
  - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
  - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
  - Instruction encodings that are unallocated.
  - Instruction encodings for instructions that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the \texttt{SCTLR_EL1 \{ITD, SED, CP15BEN\}} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by \texttt{HCR_EL2.HCD} or \texttt{SCR_EL3.HCE}.
  - An SMC instruction when disabled by \texttt{SCR_EL3.SMD}.
  - An HLT instruction when disabled by \texttt{EDSCR.HDE}.
- Attempted execution of an MSR or MRS instruction to access \texttt{SP_EL0} when the value of \texttt{SPSel.SP} is 0.
- Attempted execution, in Debug state, of:
  - A DCPS1 instruction when the value of \texttt{HCR_EL2.TGE} is 1 and EL2 is disabled or not implemented in the current Security state.
  - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
  - A DCPS3 instruction when the value of \texttt{EDSCR.SDD} is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13 mon. See ‘Traps to EL3 of monitor functionality from Secure EL1 using AArch32’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- In Debug state when the value of \texttt{EDSCR.SDD} is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
• An exception that is taken to EL2 because the value of \texttt{HCR\_EL2.TGE} is 1 that, if the value of \texttt{HCR\_EL2.TGE} was 0 would have been reported with an \texttt{ESR\_ELx.EC} value of \texttt{0b000111}.

• When SVE is not implemented, attempted execution of:
  ◦ An SVE instruction.
  ◦ An MSR or MRS instruction to access \texttt{ZCR\_EL1}, \texttt{ZCR\_EL2}, or \texttt{ZCR\_EL3}.

• In Non-transactional state, attempted execution of a TCOMMIT instruction.

### ISS encoding for an exception from a WFI or WFE instruction

<table>
<thead>
<tr>
<th>CV</th>
<th>COND</th>
<th>RES0</th>
<th>Ti</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
</tr>
</tbody>
</table>

#### CV, bit [24]
Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1.
• When a T32 instruction is trapped, it is \texttt{IMPLEMENTATION DEFINED} whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally \texttt{UNKNOWN} value.

#### COND, bits [23:20]
The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to \texttt{0b1110}.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1 and:
  ◦ If the instruction is conditional, COND is set to the condition code field value from the instruction.
  ◦ If the instruction is unconditional, COND is set to \texttt{0b1110}.
• A conditional A32 instruction that is known to pass its condition code check can be presented either:
  ◦ With COND set to \texttt{0b1110}, the value for unconditional.
  ◦ With the COND value held in the instruction.
• When a T32 instruction is trapped, it is \texttt{IMPLEMENTATION DEFINED} whether:
  ◦ CV is set to 0 and COND is set to an \texttt{UNKNOWN} value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  ◦ CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
• For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is \texttt{IMPLEMENTATION DEFINED} whether the COND field is set to \texttt{0b1110}, or to the value of any condition that applied to the instruction.

This field resets to an architecturally \texttt{UNKNOWN} value.

#### Bits [19:1]
Reserved, \texttt{RES0}. 
TI, bit [0]

Trapped instruction. Possible values of this bit are:

<table>
<thead>
<tr>
<th>TI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>WFI trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>WFE trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating this exception:

- `SCTLR_EL1`.{nTWE, nTWI}.
- `HCR_EL2`.{TWE, TWI}.
- `SCR_EL3`.{TWE, TWI}.

### ISS encoding for an exception from an MCR or MRC access

<table>
<thead>
<tr>
<th></th>
<th>CV</th>
<th>COND</th>
<th>Opc2</th>
<th>Opc1</th>
<th>CRn</th>
<th>Rt</th>
<th>CRm</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.
Opc2, bits [19:17]

The Opc2 value from the issued instruction.
For a trapped VMRS access, holds the value 0b000.
This field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.
For a trapped VMRS access, holds the value 0b111.
This field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.
For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.
This field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
This field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.
For a trapped VMRS access, holds the value 0b0000.
This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRC or VMRS instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000011:

- **CNTKCTL_EL1**, {EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **PMUSERENR_EL0**, {ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **AMUSERENR_EL0**, EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **HCR_EL2**, {TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- **HCR_EL2**, TTLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
• **HCR_EL2.** (TSW, TPC, TPU) for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **HCR_EL2.TACR** for accesses to the Auxiliary Control Register at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **HCR_EL2.TIDCP** for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **HCR_EL2.** (TID1, TID2, TID3), for accesses to ID registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **CPTR_EL2.TCPAC** for accesses to **CPACR_EL1** or **CPACR** using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **HSTR_EL2.T<n>** for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **CNTHCTL_EL2.EL1PCEN** for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **MDCR_EL2.** (TPM, TPMCR), for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **CPTR_EL2.TAM** for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **CPTR_EL2.** TCPAC for accesses to **CPACR_EL1** or **CPACR** using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **CPTR_EL3.TCPAC** for accesses to **CPACR** from EL1 and EL2, and accesses to **HCPTR** from EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.

• **HCR_EL2.TID0**, for accesses to the **JIDR** register in the ID group 0 at EL0 and EL1 using AArch32, MRC access (coproc == 0b1111) trapped to EL2.

• **CPTR_EL2.TTA**, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **MDCR_EL2.TDRA**, for accesses to Debug ROM registers **DBGDRAR** and AArch-DBGDSAR using AArch32, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **MDCR_EL2.TDOSA**, for accesses to powerdown debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

• **MDCR_EL3.TDA**, for accesses to other debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.

• **MDCR_EL3.TDOSA**, for accesses to powerdown debug registers using AArch32, MCR or MRC access (coproc == 0b1111) trapped to EL3.

• **MDCR_EL3.TDA**, for accesses to other debug registers, using AArch32, MCR or MRC access (coproc == 0b1111) trapped to EL3.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000101:

• **CPACR_EL1.TTA** for accesses to trace registers, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.

• **MDSCR_EL1.TDCC**, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.

• If ARMv8.6-FGT is implemented, **MDCR_EL2.TDCC** and **MDCR_EL3.TDCC**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.

• **HCR_EL2.TID0**, for accesses to the **FPSID** register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.

• **HCR_EL2.TID3**, for accesses to registers in ID group 3 including **MVFR0**, **MVFR1** and **MVFR2**, VMRS access trapped to EL2.

**ISS encoding for an exception from an MCRR or MRRC access**

```
<table>
<thead>
<tr>
<th>CV</th>
<th>COND</th>
<th>Opc1</th>
<th>RES0</th>
<th>Rt2</th>
<th>Rt</th>
<th>CRm</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001000:

• **HCR_EL2.TID0**, for accesses to the **FPSID** register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.

• **HCR_EL2.TID3**, for accesses to registers in ID group 3 including **MVFR0**, **MVFR1** and **MVFR2**, VMRS access trapped to EL2.
**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Opc1, bits [19:16]**

The Opc1 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

**Bit [15]**

Reserved, RES0.

**Rt2, bits [14:10]**

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.
Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

CRm, bits [4:1]

The CRm value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCRR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRRC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000100:

- **CNTKCTL_EL1**, {EL0PTEN, EL0V滕EN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **PMUSERENR_EL0**, {CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **AMUSERENR_EL0**, {EN}, for accesses to Activity Monitors registers AMEV.cnTR0<n> and AMEV.cnTR1<n> from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **HCR_EL2**, {TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **HSTR_EL2.T<n>**, for accesses to System registers using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CNTKCTL_EL2**, {EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **MDCR_EL2**, {TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL2.TAM**, for accesses to Activity Monitors registers registers AMEV.cnTR0<n> and AMEV.cnTR1<n> from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **MDACR_EL3**, {TPM, TPDC}, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL3.TAM**, for accesses to Activity Monitors registers registers AMEV.cnTR0<n> and AMEV.cnTR1<n> from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL3**, for accesses to trace registers using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- **If ARMv8.6-FGT is implemented**, HDFGRTR_EL2, PMCCNTR_EL0 for MRRC access and HDFGWTR_EL2, PMCCNTR_EL0 for MCRR access to PMCCNTR at EL0, trapped to EL2.

The following sections describe configuration settings for generating exceptions that are reported using EC value 0b001100:

- **CPACR_EL1**.TTA for accesses to trace registers using MCR or MRC instructions, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- **MDSCCR_EL1**, {TDCC, TPM}, for accesses to the Debug Communications Channel (DCC) registers DBGDSAR and DBGDRAR at EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- **CPTR_EL2.TTA**, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- **MDACR_EL2**, {TPM, TPDC}, for accesses to Debug ROM registers DBGDRAR and AArch-DBGDSAR using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- **CPTR_EL3**. TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.
• **MDCR_EL3.TDOSA**, for traps to powerdown debug registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

• **MDCR_EL3.TDA**, for accesses to other debug registers, using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

### ISS encoding for an exception from an LDC or STC instruction

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | imm8 | RES0 | Rn | Offset | AM | Direction |

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally **UNKNOWN** value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether:
  - CV is set to 0 and COND is set to an **UNKNOWN** value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is **IMPLEMENTATION DEFINED** whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally **UNKNOWN** value.

**imm8, bits [19:12]**

The immediate value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

**Bits [11:10]**

Reserved, RES0.
Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See ‘Mapping of the general-purpose registers between the Execution states’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Subtract offset.</td>
</tr>
<tr>
<td>0b1</td>
<td>Add offset.</td>
</tr>
</tbody>
</table>

This bit corresponds to the U bit in the instruction encoding.

This field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

<table>
<thead>
<tr>
<th>AM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Immediate unindexed.</td>
</tr>
<tr>
<td>0b001</td>
<td>Immediate post-indexed.</td>
</tr>
<tr>
<td>0b010</td>
<td>Immediate offset.</td>
</tr>
<tr>
<td>0b011</td>
<td>Immediate pre-indexed.</td>
</tr>
<tr>
<td>0b100</td>
<td>For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.</td>
</tr>
<tr>
<td>0b110</td>
<td>For a trapped STC instruction, this encoding is reserved.</td>
</tr>
</tbody>
</table>

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in ‘Reserved values in System and memory-mapped registers and translation table entries’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

This field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to memory. STC instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from memory. LDC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000110:

- MDSCR_EL1.TDCC, for accesses using AArch32 state, LDC access to DBGDTRTXint or STC access to DBGDTRRXint trapped to EL1 or EL2.
- MDSCR_EL2.TDA, for accesses using AArch32 state, LDC access to DBGDTRTXint or STC access to DBGDTRRXint MCR or MRC access trapped to EL2.
- MDSCR_EL3.TDA, for accesses using AArch32 state, LDC access to DBGDTRTXint or STC access to DBGDTRRXint MCR or MRC access trapped to EL3.
If ARMv8.6-FGT is implemented, MDCR_EL2.TDCC and MDCR_EL3.TDCC, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.

### ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN or CPTR_ELx.TFP

<table>
<thead>
<tr>
<th>CV</th>
<th>COND</th>
<th>RES0</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
</tbody>
</table>

The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.

For an implementation that does not include either SVE or support for floating-point and Advanced SIMD, the exception is reported using the EC value 0b000000.

#### CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

#### COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.
Bits [19:0]

Reserved, RES0.

The following sections describe the configuration settings for the traps that are reported using EC value 0b000111:

- **CPACR_EL1.FPEN**, for accesses to SIMD and floating-point registers trapped to EL1.
- **CPTR_EL2.TFP**, for accesses to SIMD and floating-point registers trapped to EL2.
- **CPTR_EL3.TFP**, for accesses to SIMD and floating-point registers trapped to EL3.

**ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [24:0]

When SVE is implemented:

Reserved, RES0.

Otherwise:

Reserved, RES0.

The accesses covered by this trap include:

- Execution of SVE instructions.
- Accesses to the SVE system registers, ZCR_ELx and ID_AA64ZFR0_EL1.

For an implementation that does not include SVE, the exception is reported using the EC value 0b000000.

**ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [24:0]

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions see ‘The Illegal Execution state exception’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile and ‘PC alignment checking’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

‘Stack pointer alignment checking’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the configuration settings for generating SP alignment fault exceptions.

**ISS encoding for an exception from HVC or SVC instruction execution**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>imm16</td>
</tr>
</tbody>
</table>

Page 518
Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.


For A64 instructions, see ‘SVC’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile and ‘HVC’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If ARMv8.6-FGT is implemented, HFGITR_EL2.{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | CCKNOWNPASS | RES0 |

For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because HCR_EL2.TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.
**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is only valid if CCKNOWPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.

**CCKNOWPASS, bit [19]**

Indicates whether the instruction might have failed its condition code check.

<table>
<thead>
<tr>
<th>CCKNOWPASS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The instruction was unconditional, or was conditional and passed its condition code check.</td>
</tr>
<tr>
<td>0b1</td>
<td>The instruction was conditional, and might have failed its condition code check.</td>
</tr>
</tbody>
</table>

**Note**

In an implementation in which an SMC instruction that fails it code check is not trapped, this field can always return the value 0.

This field resets to an architecturally UNKNOWN value.

**Bits [18:0]**

Reserved, RES0.

HCR_EL2.TSC describes the configuration settings for trapping SMC instructions to EL2.

See ‘System calls’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the case where these exceptions are trapped to EL3.

**ISS encoding for an exception from SMC instruction execution in AArch64 state**

```
  24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
  |RES0|imm16|
```

Page 520
Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

This field resets to an architecturally **UNKNOWN** value.

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

HCR_EL2.TSC describes the configuration settings for trapping SMC from EL1 modes.

‘System calls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Op0</td>
<td>Op2</td>
<td>Op1</td>
<td>CRn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

CRn, bits [13:10]

The CRn value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

This field resets to an architecturally **UNKNOWN** value.
**CRm, bits [4:1]**

The CRm value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write access, including MSR instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read access, including MRS instructions.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

For exceptions caused by System instructions, see 'System' subsection of 'Branches, exception generating and System instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for the encoding values returned by an instruction.

The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- **SCTLR_EL1.UCI**, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.UCT**, for accesses to **CTR_EL0** using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.DZE**, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.UMA**, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **CPACR_EL1.TTA**, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **MDSCR_EL1.TDCC**, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If ARMv8.6-FGT is implemented, **MDCR_EL2.TDCC** and **MDCR_EL3.TDCC**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.
- **CNTKCTL_EL1.{EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}** accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **PMUSERENR_EL0.{ER, CR, SW, EN}**, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **AMUSERENR_EL0.EN**, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **HCR_EL2.{TRVM, TVM}**, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2.TDZ**, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2.TTLB**, for execution of TLB maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2.TACP**, for accesses to **CPACR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
- **CPTR_EL2.TTCP**, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2.TTRF**, for accesses to the trace filter register, **TRFCR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2.TDRA**, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2.TDOSA**, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- **CNTKCTL_EL2.{EL1PCEN, EL1PCTEN}**, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
• **MDCR_EL2.TDA**, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.

• **MDCR_EL2. (TPM, TPMCR)**, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.

• **CPRTR_EL2.TAM**, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.

• **HCR_EL2.APK**, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL2.

• **HCR_EL2. (NV, NV1)**, for Nested virtualization register access, using AArch64 state, MSR or MRS access trapped to EL2.

• **HCR_EL2.AT**, for execution of AT S1E* instructions, using AArch64 state, MSR or MRS access trapped to EL2.

• **HCR_EL2. (TERR, FIEN)**, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL2.

• **SCR_EL3.APK**, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **SCR_EL3.ST**, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **SCR_EL3. (TERR, FIEN)**, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **CRTR_EL3.TCPAC**, for accesses to **CPRTR_EL2** and **CPACR_EL1** using AArch64 state, MSR or MRS access trapped to EL3.

• **CRTR_EL3.TTA**, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **MDCR_EL3.TTRF**, for accesses to the filter trace control registers, **TRFCR_EL1** and **TRFCR_EL2**, using AArch64 state, MSR or MRS access trapped to EL3.

• **MDCR_EL3.TDA**, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **MDCR_EL3.TDOSA**, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **MDCR_EL3.TPM**, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.

• **CPRTR_EL3.TAM**, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL3.

• If ARMv8.2-EVT is implemented, **HCR_EL2. (TTLBOS, TTLBIS, TICAB, TOCU, TID4)** and **HCR2. (TTLBIS, TICAB, TOCU, TID4)** control traps for EL1 and EL0 Cache controls that use this EC value.

• If ARMv8.6-FGT is implemented:
  - **SCR_EL3.FGTEn**, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - **HFGTR_EL2** for reads and **HFGWTR_EL2** for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
  - **HFGITR_EL2** for execution of system instructions, MSR or MRS access trapped to EL2.
  - **HDFGTR_EL2** for reads and **HDFGWTR_EL2** for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
  - **HAFGRTR_EL2** for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.

### ISS encoding for an IMPLEMENTATION DEFINED exception to EL3

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---
Bits [24:13]
Reserved, RES0.

SET, bits [12:11]
Synchronous Error Type. When the RAS Extension is implemented and IFSC is \texttt{0b010000}, describes the state of the PE after taking the Instruction Abort exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>SET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b00}</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>\texttt{0b10}</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>\texttt{0b11}</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

Note
Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the IFSC field is not \texttt{0b010000}.

This field resets to an architecturally UNKNOWN value.

FnV, bit [10]
FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>FAR is valid.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>FAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid if the IFSC code is \texttt{0b010000}. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

EA, bit [9]
External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.

Bit [8]
Reserved, RES0.

S1PTW, bit [7]
For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>
For any abort other than a stage 2 fault this bit is RES0. This field resets to an architecturally UNKNOWN value.

**Bit [6]**

Reserved, RES0.

**IFSC, bits [5:0]**

Instruction Fault Status Code. Possible values of this field are:

<table>
<thead>
<tr>
<th>IFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault, level 0 of translation or translation table base register</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 1</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 2</td>
</tr>
<tr>
<td>0b001110</td>
<td>Access flag fault, level 3</td>
</tr>
<tr>
<td>0b001111</td>
<td>Access flag fault, level 3</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 0</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk</td>
</tr>
<tr>
<td>0b011100</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 0</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM]. Otherwise reserved.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**
Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor; they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally UNKNOWN value.

**ISS encoding for an exception from a Data Abort**

<table>
<thead>
<tr>
<th>ISV</th>
<th>SAS</th>
<th>SSE</th>
<th>SRT</th>
<th>SF</th>
<th>AR</th>
<th>VNCR</th>
<th>SET</th>
<th>FnV</th>
<th>EA</th>
<th>CM</th>
<th>S1PTW</th>
<th>WnR</th>
<th>DFSC</th>
</tr>
</thead>
</table>

**ISV, bit [24]**

Instruction syndrome valid. Indicates whether the syndrome information in ISS[23:14] is valid.

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No valid instruction syndrome. ISS[23:14] are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>ISS[23:14] hold a valid instruction syndrome.</td>
</tr>
</tbody>
</table>

This bit is 0 for all faults reported in ESR_EL2 except the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback.
- AArch32 instructions where the instruction:
  - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
  - Is not performing register writeback.
  - Is not using R15 as a source or destination register.

For these cases, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

ISV is 0 for all faults reported in ESR_EL1 or ESR_EL3.

When the RAS Extension is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When the RAS Extension is not implemented, the value of ISV on a synchronous External abort on a stage 2 translation table walk is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**SAS, bits [23:22]**

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

<table>
<thead>
<tr>
<th>SAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Byte</td>
</tr>
<tr>
<td>0b01</td>
<td>Halfword</td>
</tr>
<tr>
<td>0b10</td>
<td>Word</td>
</tr>
<tr>
<td>0b11</td>
<td>Doubleword</td>
</tr>
</tbody>
</table>

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.
SSE, bit [21]

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sign-extension not required.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data item must be sign-extended.</td>
</tr>
</tbody>
</table>

For all other operations this bit is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

SRT, bits [20:16]

Syndrome Register transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction. If the exception was taken from an Exception level that is using AArch32 then this is the AArch64 view of the register. See ‘Mapping of the general-purpose registers between the Execution states’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

SF, bit [15]

Width of the register accessed by the instruction is Sixty-Four. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction loads/stores a 32-bit wide register.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction loads/stores a 64-bit wide register.</td>
</tr>
</tbody>
</table>

Note

This field specifies the register width identified by the instruction, not the Execution state.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

AR, bit [14]

Acquire/Release. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction did not have acquire/release semantics.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction did have acquire/release semantics.</td>
</tr>
</tbody>
</table>

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

VNCR, bit [13]
**When ARMv8.4-NV is implemented:**

Indicates that the fault came from use of VNCR_EL2 register by EL1 code.

<table>
<thead>
<tr>
<th>VNCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault was not generated by the use of VNCR_EL2, by an MRS or MSR instruction executed at EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault was generated by the use of VNCR_EL2, by an MRS or MSR instruction executed at EL1.</td>
</tr>
</tbody>
</table>

This field is 0 in ESR_EL1.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**SET, bits [12:11]**

Synchronous Error Type. When the RAS Extension is implemented and DFSC is 0b010000, describes the state of the PE after taking the Data Abort exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>SET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b11</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Note**

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010000.

This field resets to an architecturally UNKNOWN value.

**FnV, bit [10]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>FAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

**EA, bit [9]**

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.
**CM, bit [8]**

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DCZVA instruction is not classified as a cache maintenance instruction, and therefore its execution cannot cause this field to be set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**S1PTW, bit [7]**

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>

For any abort other than a stage 2 fault this bit is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

**WnR, bit [6]**

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort caused by an instruction reading from a memory location.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by an instruction writing to a memory location.</td>
</tr>
</tbody>
</table>

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is **UNKNOWN** for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

This field resets to an architecturally **UNKNOWN** value.

**DFSC, bits [5:0]**

Data Fault Status Code. Possible values of this field are:
<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault, level 0 of translation or translation table base register.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b001011</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001100</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Synchronous Tag Check fail</td>
</tr>
<tr>
<td>0b001110</td>
<td>Synchronous External abort, on translation table walk, level 0.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011001</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 0.</td>
</tr>
<tr>
<td>0b011011</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100001</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM]. Otherwise reserved.</td>
</tr>
<tr>
<td>0b110100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown).</td>
</tr>
<tr>
<td>0b110101</td>
<td>IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).</td>
</tr>
<tr>
<td>0b111010</td>
<td>Section Domain Fault, used only for faults reported in the PAR_EL1.</td>
</tr>
<tr>
<td>0b111110</td>
<td>Page Domain Fault, used only for faults reported in the PAR_EL1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults‘ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally UNKNOWN value.
ISS encoding for an exception from a trapped floating-point exception

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[24]</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>[23]</td>
<td>Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions. The possible values of this bit are:</td>
</tr>
<tr>
<td>[10:8]</td>
<td>For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1. For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>[7]</td>
<td>Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IDF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input denormal floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input denormal floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IXF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Inexact floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inexact floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>UFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>OFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>DZF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Divide by Zero floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Divide by Zero floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IOF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Invalid Operation floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invalid Operation floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the FPCR, `{IDE, IXE, UFE, OFE, DZE, IOE}` bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the FPSCR, `{IDE, IXE, UFE, OFE, DZE, IOE}` bits enable each of the floating-point exception traps.

**ISS encoding for an SError interrupt**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| IDS | RES0 | IESB | AET | EA | RES0 | DFSC |

**IDS, bit [24]**

IMPLEMENTATION DEFINED syndrome. Possible values of this bit are:

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bits[23:0] of the ISS field holds the fields described in this encoding.</td>
</tr>
</tbody>
</table>

**Note**

If the RAS Extension is not implemented, this means that bits[23:0] of the ISS field are RES0.

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Bits[23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt.</td>
</tr>
</tbody>
</table>

**Note**

This field was previously called ISV.

This field resets to an architecturally UNKNOWN value.

**Bits [23:14]**

Reserved, RES0.

**IESB, bit [13]**

When ARMv8.2-IESB is implemented:

Implicit error synchronization event.

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately.</td>
</tr>
<tr>
<td>0b1</td>
<td>The SError interrupt was synchronized by the implicit error synchronization event and taken immediately.</td>
</tr>
</tbody>
</table>

This field is RES0 if the value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension and ARMv8.2-IESB.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**AET, bits [12:10]**

Asynchronous Error Type.

When the RAS Extension is implemented and DFSC is 0b010001, describes the state of the PE after taking the SError interrupt exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>AET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b001</td>
<td>Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b010</td>
<td>Restartable error (UEO).</td>
</tr>
<tr>
<td>0b011</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b110</td>
<td>Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall state of the PE is reported. For example, if both a Recoverable and Unrecoverable error occurred, the state is Unrecoverable.

**Note**

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

**EA, bit [9]**

External abort type. When the RAS Extension is implemented, this bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

**Bits [8:6]**

Reserved, RES0.
**DFSC, bits [5:0]**

Data Fault Status Code. When the RAS Extension is implemented, possible values of this field are:

<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Uncategorized.</td>
</tr>
<tr>
<td>0b010001</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If the RAS Extension is not implemented, this field is RES0.

---

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

---

**ISS encoding for an exception from a Breakpoint or Vector Catch debug exception**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | IFSC |

**Bits [24:6]**

Reserved, RES0.

**IFSC, bits [5:0]**

Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

---

**ISS encoding for an exception from a Software Step exception**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ISV | RES0 | EX  | IFSC |

**ISV, bit [24]**

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EX bit is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>EX bit is valid.</td>
</tr>
</tbody>
</table>

See the EX bit description for more information.

This field resets to an architecturally UNKNOWN value.
Bits [23:7]
Reserved, RES0.

EX, bit [6]
Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

<table>
<thead>
<tr>
<th>EX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An instruction other than a Load-Exclusive instruction was stepped.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Load-Exclusive instruction was stepped.</td>
</tr>
</tbody>
</table>

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

This field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]
Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see ‘Software Step exceptions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile..

ISS encoding for an exception from a Watchpoint exception

<table>
<thead>
<tr>
<th>24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0     VNCR</td>
</tr>
</tbody>
</table>

Bits [24:14]
Reserved, RES0.

VNCR, bit [13]
When ARMv8.4-NV is implemented:

Indicates that the watchpoint came from use of VNCR_EL2 register by EL1 code.

<table>
<thead>
<tr>
<th>VNCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The watchpoint was not generated by the use of VNCR_EL2 by EL1 code.</td>
</tr>
<tr>
<td>0b1</td>
<td>The watchpoint was generated by the use of VNCR_EL2 by EL1 code.</td>
</tr>
</tbody>
</table>

This field is 0 in ESR_EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

Bits [12:9]
Reserved, RES0.
CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction. The DC ZVA instruction is not classified as a cache maintenance instruction, and therefore its execution cannot cause this field to be set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint exception caused by an instruction reading from a memory location.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint exception caused by an instruction writing to a memory location.</td>
</tr>
</tbody>
</table>

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

This field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

ISS encoding for an exception from execution of a Breakpoint instruction

<table>
<thead>
<tr>
<th>Bit</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>RES0</td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Reserved, RES0.
Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary. For the AArch32 BKPT instructions, the comment field is described as the immediate field.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see ‘Breakpoint instruction exceptions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

ISS encoding for an exception from ERET, ERETTAA or ERETAB instruction

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ERETERETA |

This EC value applies when ARMv8.6-FGT is implemented, or when HCR_EL2.NV is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2. Possible values are:

<table>
<thead>
<tr>
<th>ERET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERET instruction trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERETTAA or ERETAB instruction trapped to EL2.</td>
</tr>
</tbody>
</table>

If this bit is 0, the ERETA field is RES0.

This field resets to an architecturally UNKNOWN value.

EREETA, bit [0]

Indicates whether an ERETTAA or ERETAB instruction was trapped to EL2. Possible values are:

<table>
<thead>
<tr>
<th>ERETTAA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERETTAA instruction trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERETAB instruction trapped to EL2.</td>
</tr>
</tbody>
</table>

When the ERET field is 0, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see HCR_EL2.NV.

If ARMv8.6-FGT is implemented, HFGITR_EL2.ERET controls fine-grained trap exceptions from ERET, ERETTAA and ERETAB execution.

ISS encoding for an exception from a TSTART instruction

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Rd | RES0 |

Bits [24:10]

Reserved, RES0.

Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.
Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td>BTYPE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [24:2]

Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see 'The AArch64 application level programmers model' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section B1.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [24:0]

Reserved, RES0.

For more information about generating these exceptions, see:

- HCR_EL2.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- SCR_EL3.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for an exception from a Pointer Authentication instruction authentication failure

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td>Bit</td>
<td>Bit</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>[1]</td>
<td>[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [24:2]

Reserved, RES0.

Bit [1], bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

<table>
<thead>
<tr>
<th>Bit [1]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction Key.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data Key.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
Bit [0], bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

<table>
<thead>
<tr>
<th>Bit [0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A key</td>
</tr>
<tr>
<td>0b1</td>
<td>B key</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following instructions generate an exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTIASP, AUTIAZ, AUTIA1716.
- AUTIBSP, AUTIBZ, AUTIB1716.
- AUTIA, AUTDA, AUTIB, AUTDB.
- AUTIZA, AUTIZB, AUTDZA, AUTDZB.

It is **IMPLEMENTATION DEFINED** whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a translation fault when the address is accessed:

- RETAA, RETAB.
- BRAA, BRAB, BLRAA, BLRAB.
- BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

**Accessing the ESR_EL2**

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL2 using the mnemonic ESR_EL2 or ESR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, ESR_EL2**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return ESR_EL1;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return ESR_EL2;
elsif PSTATE.EL == EL3 then
  return ESR_EL2;
```

**MSR ESR_EL2, <Xt>**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    ESR_EL1 = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  ESR_EL2 = X[t];

MRS <Xt>, ESR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.ESR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x138];
  else
    return ESR_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return ESR_EL2;
  else
    return ESR_EL1;
elsif PSTATE.EL == EL3 then
  return ESR_EL1;

MSR ESR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ESR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x138] = X[t];
  else
    ESR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    ESR_EL2 = X[t];
  else
    ESR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  ESR_EL1 = X[t];
ESR_EL3, Exception Syndrome Register (EL3)

The ESR_EL3 characteristics are:

**Purpose**

Holds syndrome information for an exception taken to EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to ESR_EL3 are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ESR_EL3 is a 64-bit register.

**Field descriptions**

The ESR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |

When an **UNPREDICTABLE** instruction is treated as **UNDEFINED**, and the exception is taken to EL3, the value of ESR_EL3 is **UNKNOWN**. The value written to ESR_EL3 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not **UNPREDICTABLE** at that Exception level, in order to avoid the possibility of a privilege violation.

**Bits [63:32]**

Reserved, RES0.

**EC, bits [31:26]**

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:
<table>
<thead>
<tr>
<th>EC</th>
<th>Meaning</th>
<th>ISS</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Unknown reason.</td>
<td>ISS encoding for exceptions with an unknown reason</td>
<td></td>
</tr>
<tr>
<td>0b000001</td>
<td>Trapped WFI or WFE instruction execution. Conditional WFE and WFI instructions that fail their condition code check do not cause an exception.</td>
<td>ISS encoding for an exception from a WFI or WFE instruction</td>
<td></td>
</tr>
<tr>
<td>0b000011</td>
<td>Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for an exception from an MCR or MRC access</td>
<td></td>
</tr>
<tr>
<td>0b000100</td>
<td>Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for an exception from an MCRR or MRRC access</td>
<td></td>
</tr>
<tr>
<td>0b000101</td>
<td>Trapped MCR or MRC access with (coproc==0b1110).</td>
<td>ISS encoding for an exception from an MCR or MRRC access</td>
<td></td>
</tr>
<tr>
<td>0b000110</td>
<td>Trapped LDC or STC access. The only architected uses of these instructions are:</td>
<td>ISS encoding for an exception from an LDC or STC instruction</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An STC to write data to memory from DBGDTRRXint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• An LDC to read data from memory to DBGDTRTXint.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b000111</td>
<td>Access to SVE, Advanced SIMD, or floating-point functionality trapped by CPACR_EL1.FPEN, CPTR_EL2.FPEN, CPTR_EL2.TFP, or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in 'EC encodings when routing exceptions to EL2' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A</td>
<td>ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN or CPTR_ELx.TFP</td>
<td></td>
</tr>
</tbody>
</table>
architecture profile, section D1.10.4.

0b001001 Trapped use of a Pointer authentication instruction because HCR_EL2.API == 0 || SCR_EL3.API == 0.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0.

When ARMv8.3-PAuth is implemented

0b001100 Trapped MRRC access with (coproc==0b1110).

ISS encoding for an exception from an MCRR or MRRC access.

When ARMv8.5-BTI is implemented

0b001101 Branch Target Exception.

ISS encoding for an exception from Branch Target Identification instruction.

0b001110 Illegal Execution state.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault.

0b010011 SMC instruction execution in AArch32 state, when SMC is not disabled. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1.

ISS encoding for an exception from SMC instruction execution in AArch32 state.

0b010101 SVC instruction execution in AArch64 state.

ISS encoding for an exception from HVC or SVC instruction execution.

0b010110 HVC instruction execution in AArch64 state, when HVC is not disabled.

ISS encoding for an exception from HVC or SVC instruction execution.

0b010111 SMC instruction execution in AArch64 state, when SMC is not disabled. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1.

ISS encoding for an exception from SMC instruction execution in AArch64 state.

0b011000 Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b000000, 0b000001 or 0b000111.

If ARMv8.4-IDST is implemented, also exceptions generated by a read access to the feature ID space.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state.
If ARMv8.0-CSV2 is implemented, also Cache Speculation Variant exceptions. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section C5.2.2, except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111.

**0b0110001**
Access to SVE functionality trapped as a result of CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC 0b000000. This EC is defined only if SVE is implemented.

**0b011011**
Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when SCTLR_EL1.TME == 0, at EL2 when SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0.

**0b011100**
Exception from a Pointer Authentication instruction authentication failure

**0b011111**
IMPLEMENTATION DEFINED exception to EL3.

**0b100000**
Instruction Abort from a lower Exception level, that might be using AArch32 or AArch64.

Used for MMU faults generated by instruction accesses

**ISS encoding for an exception from a SVE access**

**ISS encoding for an exception from a TSTART instruction**

**ISS encoding for an exception from a Pointer Authentication instruction authentication failure**

**ISS encoding for an IMPLEMENTATION DEFINED exception to EL3**

**ISS encoding for an Instruction Abort**

When TME is implemented

When ARMv8.3-FPAC is implemented
and synchronous
External aborts, including
synchronous parity
or ECC errors. Not
used for debug
related exceptions.

0b100001  Instruction Abort
taken without a change in Exception
level. Used for MMU faults generated by
instruction accesses and synchronous
External aborts, including
synchronous parity
or ECC errors. Not
used for debug
related exceptions.

0b100010  PC alignment fault exception.

0b100100  Data Abort from a lower Exception
level, that might be using AArch32 or
AArch64. Used for MMU faults generated by
data accesses, alignment faults
other than those caused by Stack
Pointer misalignment, and synchronous
External aborts, including
synchronous parity
or ECC errors. Not
used for debug
related exceptions.

0b100101  Data Abort taken without a change in
Exception level. Used for MMU faults generated by
data accesses, alignment faults
other than those caused by Stack
Pointer misalignment, and synchronous
External aborts, including
synchronous parity
or ECC errors. Not
used for debug
related exceptions.

0b100110  SP alignment fault exception.
Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is implementation defined.

0b101111 SError interrupt.

0b111100 BRK instruction execution in AArch64 state. This is reported in ESR_EL3 only if a BRK instruction is executed.

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is constrained unpredictable, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to an architecturally unknown value.

**IL, bit [25]**

Instruction Length for synchronous exceptions. Possible values of this bit are:

<table>
<thead>
<tr>
<th>IL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>16-bit instruction trapped.</td>
</tr>
</tbody>
</table>
| 0b1 | 32-bit instruction trapped. This value is also used when the exception is one of the following:  
  - An SError interrupt.  
  - An Instruction Abort exception.  
  - A PC alignment fault exception.  
  - An SP alignment fault exception.  
  - A Data Abort exception for which the value of the ISV bit is 0.  
  - An Illegal Execution state exception.  
  - Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning:  
    - 0b0: 16-bit T32 BKPT instruction.  
    - 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction.  
  - An exception reported using EC value 0b000000. |

This field resets to an architecturally unknown value.
ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number. For an exception taken from AArch32 state, 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value 0b1111.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
  - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
  - The value 0b1111.

When the EC field is 0b000000, indicating an exception with an unknown reason, the ISS field is not valid, RES0.

ISS encoding for exceptions with an unknown reason

<table>
<thead>
<tr>
<th>24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>reserv0</td>
</tr>
</tbody>
</table>

Bits [24:0]

Reserved, RES0.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
  - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
  - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
  - Instruction encodings that are unallocated.
  - Instruction encodings for instructions that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR_EL1, {ITD, SED, CP15BEN} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by HCR_EL2.HCD or SCR_EL3.HCE.
  - An SMC instruction when disabled by SCR_EL3.SMD.
  - An HLT instruction when disabled by EDSCR.HDE.
- Attempted execution of an MSR or MRS instruction to access SP_EL0 when the value of SPSEL.SP is 0.
- Attempted execution, in Debug state, of:
  - A DCPS1 instruction when the value of HCR_EL2.TGE is 1 and EL2 is disabled or not implemented in the current Security state.
  - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
  - A DCPS3 instruction when the value of EDSCR.SDD is 1, or when EL3 is not implemented.
• When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon. See 'Traps to EL3 of monitor functionality from Secure EL1 using AArch32' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
• In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
• In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
• An exception that is taken to EL2 because the value of HCR_EL2.TGE is 1 that, if the value of HCR_EL2.TGE was 0 would have been reported with an ESR_ELx.EC value of 0b000111.
• When SVE is not implemented, attempted execution of:
  ◦ An SVE instruction.
  ◦ An MSR or MRS instruction to access ZCR_EL1, ZCR_EL2, or ZCR_EL3.
• In Non-transactional state, attempted execution of a TCOMMIT instruction.

**ISS encoding for an exception from a WFI or WFE instruction**

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>CV</td>
<td>COND</td>
<td>RES0</td>
<td>Ti</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1.
• When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1 and:
  ◦ If the instruction is conditional, COND is set to the condition code field value from the instruction.
  ◦ If the instruction is unconditional, COND is set to 0b1110.
• A conditional A32 instruction that is known to pass its condition code check can be presented either:
  ◦ With COND set to 0b1110, the value for unconditional.
  ◦ With the COND value held in the instruction.
• When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  ◦ CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  ◦ CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
• For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.
**Bits [19:1]**

Reserved, RES0.

**TI, bit [0]**

Trapped instruction. Possible values of this bit are:

<table>
<thead>
<tr>
<th>TI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>WFI trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>WFE trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following fields describe configuration settings for generating this exception:

- **SCTLR_EL1**: \{nTWE, nTWI\}.
- **HCR_EL2**: \{TWE, TWI\}.
- **SCR_EL3**: \{TWE, TWI\}.

**ISS encoding for an exception from an MCR or MRC access**

```
   24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
  CV COND  Opc2  Opc1  CRn  Rt  CRm  Direction
```

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally **UNKNOWN** value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether:
  - CV is set to 0 and COND is set to an **UNKNOWN** value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean
that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Opc2, bits [19:17]**

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

This field resets to an architecturally UNKNOWN value.

**Opc1, bits [16:14]**

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value 0b111.

This field resets to an architecturally UNKNOWN value.

**CRn, bits [13:10]**

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

This field resets to an architecturally UNKNOWN value.

**Rt, bits [9:5]**

The Rt value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**CRm, bits [4:1]**

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

This field resets to an architecturally UNKNOWN value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRC or VMRS instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000011:

- **CNTKCTL_EL1** (EL0PTEN, EL0VTEN, EL0PCTEN, ELOVCTEN), for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **PMUSERENR_EL0** (ER, CR, SW, EN), for accesses to Performance Monitor registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000101:

- **CPACR_EL1.TTA**, for accesses to trace registers, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- **MDCR_EL1.TDCC**, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- If ARMv8.6-FGT is implemented, **MDCR_EL2.TDCC** and **MDCR_EL3.TDCC**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.
- **HCR_EL2.TID0**, for accesses to the **IIDR** register in the ID group 0 at EL0 and EL1 using AArch32, MRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL2.TTA**, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- **MDCR_EL2.TDRA**, for accesses to Debug ROM registers **DBGDRAR** and AArch-DBGDSAR using AArch32, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- **MDCR_EL2.TDOSA**, for accesses to powerdown debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- **MDCR_EL2.TDA**, for accesses to other debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- **CPTR_EL3.TTA**, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- **MDCR_EL3.TDOSA**, for accesses to powerdown debug registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- **MDCR_EL3.TDA**, for accesses to other debug registers, using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001000:

- **HCR_EL2.TID0**, for accesses to the **FPSID** register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- **HCR_EL2.TID3**, for accesses to registers in ID group 3 including **MVFR0**, **MVFR1** and **MVFR2**, VMRS access trapped to EL2.
ISS encoding for an exception from an MCRR or MRRC access

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | Opc1 | RES0 | Rt2 | Rt | CRm | Direction |

CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.
Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See ‘Mapping of the general-purpose registers between the Execution states’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See ‘Mapping of the general-purpose registers between the Execution states’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

CRm, bits [4:1]

The CRm value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Direction, bit [0]

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCRR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRRC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following fields describe configuration settings for generating exceptions that are reported using EC value **0b000100**:

- **CNTKCTL_EL1**, {EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **PMUSERENR_EL0**, {CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **AMUSERENR_EL0**, {EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- **HCR_EL2**, (TRVM, TVM), for accesses to virtual memory control registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **HSTR_EL2**, T<n>, for accesses to System registers using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CNTHCTL_EL2**, {EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **MDCR_EL2**, (TPM, TPMCR), for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **CPTER_EL2**, TAM, for accesses to Activity Monitors registers registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- **HDFGRTR_EL2**, for MRRC access and HDFGWTR_EL2, PMCCNTR_EL0 for MCRR access to PMCCNTR at EL0, trapped to EL2.

The following sections describe configuration settings for generating exceptions that are reported using EC value **0b0001010**:

- **CPACR_EL1**, TTA for accesses to trace registers using MCR or MRC instructions, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
• MDSCR_EL1.TDCC, for accesses to the Debug Communications Channel (DCC) registers DBGDSAR and DBGDRAR at EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
• CPTR_EL2.TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
• MDSCR_EL2.TDRA, for accesses to Debug ROM registers DBGDRAR and AArch-DBGDSAR using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
• CPTR_EL3.TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.
• MDSCR_EL3.TDOSA, for traps to powerdown debug registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.
• MDSCR_EL3.TDA, for accesses to other debug registers, using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

**ISS encoding for an exception from an LDC or STC instruction**

<table>
<thead>
<tr>
<th>CV</th>
<th>COND</th>
<th>imm8</th>
<th>RES0</th>
<th>Rn</th>
<th>Offset</th>
<th>AM</th>
<th>Direction</th>
</tr>
</thead>
</table>

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1.
• When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

• When an A32 instruction is trapped, CV is set to 1 and:
  ◦ If the instruction is conditional, COND is set to the condition code field value from the instruction.
  ◦ If the instruction is unconditional, COND is set to 0b1110.
• A conditional A32 instruction that is known to pass its condition code check can be presented either:
  ◦ With COND set to 0b1110, the value for unconditional.
  ◦ With the COND value held in the instruction.
• When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
  ◦ CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  ◦ CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
• For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.
imm8, bits [19:12]

The immediate value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

**Bits [11:10]**

Reserved, **RES0**.

**Rn, bits [9:5]**

The Rn value from the issued instruction, the general-purpose register used for the transfer. The reported value gives the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is **UNKNOWN**.

This field resets to an architecturally **UNKNOWN** value.

**Offset, bit [4]**

Indicates whether the offset is added or subtracted:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Subtract offset.</td>
</tr>
<tr>
<td>0b1</td>
<td>Add offset.</td>
</tr>
</tbody>
</table>

This bit corresponds to the U bit in the instruction encoding.

This field resets to an architecturally **UNKNOWN** value.

**AM, bits [3:1]**

Addressing mode. The permitted values of this field are:

<table>
<thead>
<tr>
<th>AM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Immediate unindexed.</td>
</tr>
<tr>
<td>0b001</td>
<td>Immediate post-indexed.</td>
</tr>
<tr>
<td>0b010</td>
<td>Immediate offset.</td>
</tr>
<tr>
<td>0b011</td>
<td>Immediate pre-indexed.</td>
</tr>
<tr>
<td>0b100</td>
<td>For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved.</td>
</tr>
<tr>
<td>0b110</td>
<td>For a trapped STC instruction, this encoding is reserved.</td>
</tr>
</tbody>
</table>

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is **CONSTRAINED UNPREDICTABLE**, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to memory. STC instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from memory. LDC instruction.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally unknown value.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000110:

- **MDSCR_EL1.TDCC**, for accesses using AArch32 state, LDC access to **DBGDTRTXint** or STC access to **DBGDTRRXint** trapped to EL1 or EL2.
- **MDCR_EL2.TDA**, for accesses using AArch32 state, LDC access to **DBGDTRTXint** or STC access to **DBGDTRRXint** MCR or MRC access trapped to EL2.
- **MDCR_EL3.TDA**, for accesses using AArch32 state, LDC access to **DBGDTRTXint** or STC access to **DBGDTRRXint** MCR or MRC access trapped to EL3.
- If ARMv8.6-FGT is implemented, **MDCR_EL2.TDCC** and **MDCR_EL3.TDCC**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.

### ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from CPACR_EL1.FPEN, CPTR_EL2.FPEN or CPTR_ELx.TFP

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | RES0 |

The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.

For an implementation that does not include either SVE or support for floating-point and Advanced SIMD, the exception is reported using the EC value 0b000000.

### CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is implementation defined whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally unknown value.

### COND, bits [23:20]

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is implementation defined whether:
  - CV is set to 0 and COND is set to an unknown value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Bits [19:0]**

Reserved, RES0.

The following sections describe the configuration settings for the traps that are reported using EC value 0b000111:

- **CPACR_EL1.FPEN**, for accesses to SIMD and floating-point registers trapped to EL1.
- **CPTR_EL2.TFP**, for accesses to SIMD and floating-point registers trapped to EL2.
- **CPTR_EL2.TFP**, for accesses to SIMD and floating-point registers trapped to EL3.

**ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ**

```
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**Bits [24:0]**

When SVE is implemented:

Reserved, RES0.

Otherwise:

Reserved, RES0.

The accesses covered by this trap include:

- Execution of SVE instructions.
- Accesses to the SVE system registers, ZCR_ELx and ID_AA64ZFR0_EL1.

For an implementation that does not include SVE, the exception is reported using the EC value 0b000000.

**ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault**

```
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**Bits [24:0]**

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions see 'The Illegal Execution state exception' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile and 'PC alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

'Stack pointer alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the configuration settings for generating SP alignment fault exceptions.
ISS encoding for an exception from HVC or SVC instruction execution

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | imm16 |

Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.


If ARMv8.6-FGT is implemented, HFGITR_EL2, {SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | CKNOWNPASS | RES0 |

For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because HCR_EL2.TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally **UNKNOWN** value.

**COND, bits [23:20]**

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch64, this field is set to 0b1110.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
  - If the instruction is conditional, COND is set to the condition code field value from the instruction.
  - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
  - With COND set to 0b1110, the value for unconditional.
  - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether:
  - CV is set to 0 and COND is set to an **UNKNOWN** value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
  - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is **IMPLEMENTATION DEFINED** whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is only valid if CCKNOWNPASS is 1, otherwise it is RES0.

This field resets to an architecturally **UNKNOWN** value.

**CCKNOWNPASS, bit [19]**

Indicates whether the instruction might have failed its condition code check.

<table>
<thead>
<tr>
<th>CCKNOWNPASS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The instruction was unconditional, or was conditional and passed its condition code check.</td>
</tr>
<tr>
<td>0b1</td>
<td>The instruction was conditional, and might have failed its condition code check.</td>
</tr>
</tbody>
</table>

**Note**

In an implementation in which an SMC instruction that fails its code check is not trapped, this field can always return the value 0.

This field resets to an architecturally **UNKNOWN** value.

**Bits [18:0]**

Reserved, RES0.

HCR_EL2.TSC describes the configuration settings for trapping SMC instructions to EL2.

See ‘System calls’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the case where these exceptions are trapped to EL3.

**ISS encoding for an exception from SMC instruction execution in AArch64 state**
Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

This field resets to an architecturally **UNKNOWN** value.

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

HCR_EL2.TSC describes the configuration settings for trapping SMC from EL1 modes.

'System calls' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the case where these exceptions are trapped to EL3.

**ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Op0 | Op2 | Op1 | CRn | Rt | CRm | Direction |

Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

CRn, bits [13:10]

The CRn value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.
This field resets to an architecturally **UNKNOWN** value.

**CRm, bits [4:1]**

The CRm value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write access, including MSR instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read access, including MRS instructions.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

For exceptions caused by System instructions, see 'System' subsection of 'Branches, exception generating and System instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for the encoding values returned by an instruction.

The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- **SCTLR_EL1.uci**, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.uct**, for accesses to **CTR_EL0** using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.dze**, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **SCTLR_EL1.uma**, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **CPACR_EL1.tta**, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **MDSCR_EL1.tdcc**, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **MDSCR_EL1.TDCC**, for accesses to the Trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **CPACR_EL1.tta**, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **MDSCR_EL1.TDCC**, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If ARMv8.6-FGT is implemented, **MDCR_EL2.tdcc** and **MDCR_EL3.tdcc**, for accesses to the DCC registers at EL0 and EL1, trapped to EL2.
- **CNTKCTL_EL1.EL0PTEN, EL0VTEN, EL0PCTEN, EL0VCTEN**, accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **PMUSERENR_EL0.ER, CR, SW, EN**, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **AMUSERENR_EL0.EN**, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **HCR_EL2.TRVM, TVM**, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- **HCR_EL2.TSTF, TPC, TPU**, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2.TACR**, for accesses to the Auxiliary Control Register, **ACTLR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2.TDACP, for accesses to the auxiliary control registers, using AArch64 state, MSR or MRS access trapped to EL2.**
- **HCR_EL2.TSTF, TPC, TPU**, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2.TID1, TID2, TID3**, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **CPTR_EL2.TCPAC**, for accesses to **CPACR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
- **CPTR_EL2.TTA**, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2.TTRF**, for accesses to the trace filter register, **TRFCR_EL1**, using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2.TDRA**, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
ESR_EL3, Exception Syndrome Register (EL3)

- **MDCR_EL2**.TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- **CNTHCTL_EL2**. {EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2**.TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- **MDCR_EL2**. {TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **CPTR_EL2**.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2**.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2**. {NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2**.AT, for execution of AT S1E* instructions, using AArch64 state, MSR or MRS access trapped to EL2.
- **HCR_EL2**. {TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL2.
- **SCR_EL3**.APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **SCR_EL3**.ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **SCR_EL3**. {TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **CPTR_EL3**.TCPAC, for accesses to **CPTR_EL2** and **CPACR_EL1** using AArch64 state, MSR or MRS access trapped to EL3.
- **CPTR_EL3**.TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **MDCR_EL3**.TTRF, for accesses to the filter trace control registers, **TRFCR_EL1** and **TRFCR_EL2**, using AArch64 state, MSR or MRS access trapped to EL3.
- **MDCR_EL3**.TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **MDCR_EL3**.TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **MDCR_EL3**.TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- **CPTR_EL3**.TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL3.
- If ARMv8.2-EVT is implemented, **HCR_EL2**. {TTLBOS, TTLBIS, TICAB, TOCU, TID4} and **HCR2**. {TTLBIS, TICAB, TOCU, TID4} control traps for EL1 and EL0 Cache controls that use this EC value.
- If ARMv8.6-FGT is implemented:
  - **SCR_EL3**.FGTen, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
  - **HFGTR_EL2** for reads and **HFGWTR_EL2** for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
  - **HFGITR_EL2** for execution of system instructions, MSR or MRS access trapped to EL2.
  - **HDFGTR_EL2** for reads and **HDFGWTR_EL2** for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
  - **HAFGTR_EL2** for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.

### ISS encoding for an IMPLEMENTATION DEFINED exception to EL3

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED |

### IMPLEMENTATION DEFINED, bits [24:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

### ISS encoding for an exception from an Instruction Abort
Bits [24:13]

Reserved, RES0.

SET, bits [12:11]

Synchronous Error Type. When the RAS Extension is implemented and IFSC is 0b010000, describes the state of the PE after taking the Instruction Abort exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>SET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b11</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the IFSC field is not 0b010000.

This field resets to an architecturally UNKNOWN value.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>FAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid if the IFSC code is 0b010000. It is RES0 for all other aborts.

This field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:
For any abort other than a stage 2 fault this bit is RES0.

This field resets to an architecturally UNKNOWN value.

**Bit [6]**

Reserved, RES0.

**IFSC, bits [5:0]**

Instruction Fault Status Code. Possible values of this field are:

<table>
<thead>
<tr>
<th>IFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault, level 0 of translation or translation table base register</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 1</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 2</td>
</tr>
<tr>
<td>0b001011</td>
<td>Access flag fault, level 3</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 1</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 2</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 0</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk</td>
</tr>
<tr>
<td>0b011100</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 0</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM. Otherwise reserved.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

**Note**

Armv8.2 requires the implementation of the RAS Extension.
For more information about the lookup level associated with a fault, see 'The level associated with MMU faults’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

---

**Note**

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally **UNKNOWN** value.

---

**ISS encoding for an exception from a Data Abort**

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When the RAS Extension is not implemented, the value of ISV on a synchronous External abort on a stage 2 translation table walk is **IMPLEMENTATION DEFINED**.

This field resets to an architecturally **UNKNOWN** value.

---

**ISS, bit [24]**

Instruction syndrome valid. Indicates whether the syndrome information in ISS[23:14] is valid.

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No valid instruction syndrome. ISS[23:14] are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>ISS[23:14] hold a valid instruction syndrome.</td>
</tr>
</tbody>
</table>

This bit is 0 for all faults reported in ESR_EL2 except the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback.
- AArch32 instructions where the instruction:
  - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
  - Is not performing register writeback.
  - Is not using R15 as a source or destination register.

For these cases, ISV is **UNKNOWN** if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

ISV is 0 for all faults reported in ESR_EL1 or ESR_EL3.

When the RAS Extension is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When the RAS Extension is not implemented, the value of ISV on a synchronous External abort on a stage 2 translation table walk is **IMPLEMENTATION DEFINED**.

This field resets to an architecturally **UNKNOWN** value.

---

**SAS, bits [23:22]**

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

<table>
<thead>
<tr>
<th>SAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Byte</td>
</tr>
<tr>
<td>0b01</td>
<td>Halfword</td>
</tr>
<tr>
<td>0b10</td>
<td>Word</td>
</tr>
<tr>
<td>0b11</td>
<td>Doubleword</td>
</tr>
</tbody>
</table>

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is RES0 when the value of ISV is 0.

---
This field resets to an architecturally **UNKNOWN** value.

**SSE, bit [21]**

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sign-extension not required.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data item must be sign-extended.</td>
</tr>
</tbody>
</table>

For all other operations this bit is 0.

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**SRT, bits [20:16]**

Syndrome Register transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction. If the exception was taken from an Exception level that is using AArch32 then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**SF, bit [15]**

Width of the register accessed by the instruction is Sixty-Four. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction loads/stores a 32-bit wide register.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction loads/stores a 64-bit wide register.</td>
</tr>
</tbody>
</table>

**Note**

This field specifies the register width identified by the instruction, not the Execution state.

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**AR, bit [14]**

Acquire/Release. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction did not have acquire/release semantics.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction did have acquire/release semantics.</td>
</tr>
</tbody>
</table>

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.
**VNCR, bit [13]**

When ARMv8.4-NV is implemented:

Indicates that the fault came from use of `VNCR_EL2` register by EL1 code.

<table>
<thead>
<tr>
<th>VNCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault was not generated by the use of <code>VNCR_EL2</code>, by an MRS or MSR instruction executed at EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault was generated by the use of <code>VNCR_EL2</code>, by an MRS or MSR instruction executed at EL1.</td>
</tr>
</tbody>
</table>

This field is 0 in ESR_EL1.

This field resets to an architecturally `UNKNOWN` value.

**Otherwise:**

Reserved, `RES0`.

**SET, bits [12:11]**

Synchronous Error Type. When the RAS Extension is implemented and DFSC is 0b010000, describes the state of the PE after taking the Data Abort exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>SET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b11</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Note**

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in an unrecoverable PE state.

This field is `RES0` if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010000.

This field resets to an architecturally `UNKNOWN` value.

**FnV, bit [10]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>FAR is not valid, and holds an <code>UNKNOWN</code> value.</td>
</tr>
</tbody>
</table>

This field is valid only if the DFSC code is 0b010000. It is `RES0` for all other aborts.

This field resets to an architecturally `UNKNOWN` value.

**EA, bit [9]**

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.
This field resets to an architecturally **UNKNOWN** value.

### CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA instruction is not classified as a cache maintenance instruction, and therefore its execution cannot cause this field to be set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>

For any abort other than a stage 2 fault this bit is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

### WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort caused by an instruction reading from a memory location.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by an instruction writing to a memory location.</td>
</tr>
</tbody>
</table>

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is **UNKNOWN** for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

This field resets to an architecturally **UNKNOWN** value.

### DFSC, bits [5:0]

Data Fault Status Code. Possible values of this field are:
<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault, level 0 of translation or translation table base register.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b001100</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b010001</td>
<td>Synchronous Tag Check fail.</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 0.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011100</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 0.</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100001</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault, if the implementation includes ARMv8.1-TTHM]. Otherwise reserved.</td>
</tr>
<tr>
<td>0b110100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown).</td>
</tr>
<tr>
<td>0b110101</td>
<td>IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access).</td>
</tr>
<tr>
<td>0b111101</td>
<td>Section Domain Fault, used only for faults reported in the PAR_EL1.</td>
</tr>
<tr>
<td>0b111110</td>
<td>Page Domain Fault, used only for faults reported in the PAR_EL1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

Because Access flag faults and Permission faults can only result from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally **UNKNOWN** value.
ISS encoding for an exception from a trapped floating-point exception

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions. The possible values of this bit are:

<table>
<thead>
<tr>
<th>TFV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN.</td>
</tr>
<tr>
<td>0b1</td>
<td>One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information see 'Floating-point exception traps' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating point exception from a vector instruction.

**Note**

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from a vector instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

This field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IDF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input denormal floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input denormal floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IXF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Inexact floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inexact floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>UFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>OFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>DZF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Divide by Zero floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Divide by Zero floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

<table>
<thead>
<tr>
<th>IOF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Invalid Operation floating-point exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invalid Operation floating-point exception occurred during execution of the reported instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the `FPSCR.{IDE, IXE, UFE, OFE, DZE, IOE}` bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the `FPSCR.{IDE, IXE, UFE, OFE, DZE, IOE}` bits enable each of the floating-point exception traps.

### ISS encoding for an SError interrupt

![Table of ISS encoding](image)

**IDS, bit [24]**

**IMPLEMENTATION DEFINED** syndrome. Possible values of this bit are:

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bits[23:0] of the ISS field holds the fields described in this encoding.</td>
</tr>
</tbody>
</table>

**Note**

If the RAS Extension is not implemented, this means that bits[23:0] of the ISS field are RES0.

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Bits[23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt.</td>
</tr>
</tbody>
</table>

**Note**

This field was previously called ISV.

This field resets to an architecturally UNKNOWN value.

**Bits [23:14]**

Reserved, RES0.

**IESB, bit [13]**

**When ARMv8.2-IESB is implemented:**

Implicit error synchronization event.

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately.</td>
</tr>
<tr>
<td>0b1</td>
<td>The SError interrupt was synchronized by the implicit error synchronization event and taken immediately.</td>
</tr>
</tbody>
</table>

This field is RES0 if the value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension and ARMv8.2-IESB.

This field resets to an architecturally UNKNOWN value.
Reserved, RES0.

**AET, bits [12:10]**

Asynchronous Error Type.

When the RAS Extension is implemented and DFSC is 0b010001, describes the state of the PE after taking the SError interrupt exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>AET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b001</td>
<td>Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b010</td>
<td>Restartable error (UEO).</td>
</tr>
<tr>
<td>0b011</td>
<td>Recoverable error (UER).</td>
</tr>
<tr>
<td>0b110</td>
<td>Corrected error (CE).</td>
</tr>
</tbody>
</table>

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall state of the PE is reported. For example, if both a Recoverable and Unrecoverable error occurred, the state is Unrecoverable.

**Note**

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

**EA, bit [9]**

External abort type. When the RAS Extension is implemented, this bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field is RES0 if either:

- The RAS Extension is not implemented.
- The value returned in the DFSC field is not 0b010001.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally UNKNOWN value.

**Bits [8:6]**

Reserved, RES0.
DFSC, bits [5:0]

Data Fault Status Code. When the RAS Extension is implemented, possible values of this field are:

<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>Uncategorized.</td>
</tr>
<tr>
<td>001001</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If the RAS Extension is not implemented, this field is RES0.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally **UNKNOWN** value.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | IFSC |

**Bits [24:6]**

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally **UNKNOWN** value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

ISS encoding for an exception from a Software Step exception

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ISV | RES0 | EX | IFSC |

**ISV, bit [24]**

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EX bit is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>EX bit is valid.</td>
</tr>
</tbody>
</table>

See the EX bit description for more information.

This field resets to an architecturally **UNKNOWN** value.
Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

<table>
<thead>
<tr>
<th>EX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An instruction other than a Load-Exclusive instruction was stepped.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Load-Exclusive instruction was stepped.</td>
</tr>
</tbody>
</table>

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

This field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]

Instruction Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Software Step exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

ISS encoding for an exception from a Watchpoint exception

| 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| VNCR| RES0| CM  | RES0| WnR | DFSC|

Bits [24:14]

Reserved, RES0.

VNCR, bit [13]

When ARMv8.4-NV is implemented:

Indicates that the watchpoint came from use of VNCR_EL2 register by EL1 code.

<table>
<thead>
<tr>
<th>VNCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The watchpoint was not generated by the use of VNCR_EL2 by EL1 code.</td>
</tr>
<tr>
<td>0b1</td>
<td>The watchpoint was generated by the use of VNCR_EL2 by EL1 code.</td>
</tr>
</tbody>
</table>

This field is 0 in ESR_EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.
CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Watchpoint exception was not generated by the execution of one of</td>
</tr>
<tr>
<td></td>
<td>the System instructions identified in the description of value 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Watchpoint exception was generated by either the execution of a</td>
</tr>
<tr>
<td></td>
<td>cache maintenance instruction or by a synchronous Watchpoint exception</td>
</tr>
<tr>
<td></td>
<td>on the execution of an address translation instruction. The <strong>DC,ZVA</strong></td>
</tr>
<tr>
<td></td>
<td>instruction is not classified as a cache maintenance instruction, and</td>
</tr>
<tr>
<td></td>
<td>therefore its execution cannot cause this field to be set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Bit [7]

Reserved, **RES0**.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint exception caused by an instruction reading from a memory</td>
</tr>
<tr>
<td></td>
<td>location.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint exception caused by an instruction writing to a memory</td>
</tr>
<tr>
<td></td>
<td>location.</td>
</tr>
</tbody>
</table>

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is **UNPREDICTABLE** which watchpoint generates the Watchpoint exception.

This field resets to an architecturally **UNKNOWN** value.

DFSC, bits [5:0]

Data Fault Status Code. This field is set to 0b100010, to indicate a Debug exception.

This field resets to an architecturally **UNKNOWN** value.

For more information about generating these exceptions, see ‘Watchpoint exceptions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

ISS encoding for an exception from execution of a Breakpoint instruction

| 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

Comment

<table>
<thead>
<tr>
<th>Bits [24:16]</th>
</tr>
</thead>
</table>

Reserved, **RES0**.
Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary. For the AArch32 BKPT instructions, the comment field is described as the immediate field.

This field resets to an architecturally unknown value.

For more information about generating these exceptions, see ‘Breakpoint instruction exceptions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

ISS encoding for an exception from ERET, ERETTA or ERETAB instruction

This EC value applies when ARMv8.6-FGT is implemented, or when HCR_EL2.NV is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2. Possible values are:

<table>
<thead>
<tr>
<th>ERET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERET instruction trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERETTA or ERETAB instruction trapped to EL2.</td>
</tr>
</tbody>
</table>

If this bit is 0, the ERETA field is RES0.

This field resets to an architecturally unknown value.

ERETA, bit [0]

Indicates whether an ERETTA or ERETAB instruction was trapped to EL2. Possible values are:

<table>
<thead>
<tr>
<th>ERETA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERETTA instruction trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERETAB instruction trapped to EL2.</td>
</tr>
</tbody>
</table>

When the ERET field is 0, this bit is RES0.

This field resets to an architecturally unknown value.

For more information about generating these exceptions, see HCR_EL2.NV.

If ARMv8.6-FGT is implemented, HFGITR_EL2.ERET controls fine-grained trap exceptions from ERET, ERETTA and ERETAB execution.

ISS encoding for an exception from a TSTART instruction

Bits [24:10]

Reserved, RES0.

Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.
Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction

<table>
<thead>
<tr>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BTYPE</td>
</tr>
</tbody>
</table>

Bits [24:2]

Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see 'The AArch64 application level programmers model' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section B1.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

Bits [24:0]

Reserved, RES0.

For more information about generating these exceptions, see:

- HCR_EL2.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- SCR_EL3.API, for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for an exception from a Pointer Authentication instruction authentication failure

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Bit [1] Bit [0] |

Bits [24:2]

Reserved, RES0.

Bit [1], bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

<table>
<thead>
<tr>
<th>Bit [1]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction Key.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data Key.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
**Bit [0], bit [0]**

This field indicates whether the exception is as a result of an A key or a B key.

<table>
<thead>
<tr>
<th>Bit [0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A key.</td>
</tr>
<tr>
<td>0b1</td>
<td>B key.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following instructions generate an exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTIASP, AUTIAZ, AUTIA1716.
- AUTIBSP, AUTIBZ, AUTIB1716.
- AUTIA, AUTDA, AUTIB, AUTDB.
- AUTIZA, AUTIZB, AUTDZA, AUTDZB.

It is **IMPLEMENTATION DEFINED** whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a translation fault when the address is accessed:

- RETAA, RETAB.
- BRAA, BRAB, BLRAA, BLRAB.
- BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETTA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

**Accessing the ESR_EL3**

Accesses to this register use the following encodings:

**MRS <Xt>, ESR_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
else PSTATE.EL == EL3 then
    return ESR_EL3;

**MSR ESR_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
else PSTATE.EL == EL3 then
    ESR_EL3 = X[t];
The FAR_EL1 characteristics are:

**Purpose**

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL1.

**Configuration**

AArch64 System register FAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register `DFAR[31:0] (NS)`.

AArch64 System register FAR_EL1 bits [63:32] are architecturally mapped to AArch32 System register `IFAR[31:0] (NS)`.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

FAR_EL1 is a 64-bit register.

**Field descriptions**

The FAR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-3</td>
<td>Faulting Virtual Address for synchronous exceptions taken to EL1</td>
</tr>
<tr>
<td>31-0</td>
<td>Faulting Virtual Address for synchronous exceptions taken to EL1</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Faulting Virtual Address for synchronous exceptions taken to EL1. Exceptions that set the FAR_EL1 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), PC alignment faults (EC 0x22), and Watchpoints (EC 0x34 or 0x35). `ESR_EL1`.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL1 are **UNKNOWN**.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if `ESR_EL1`.FnV is 0, and the FAR_EL1 is **UNKNOWN** if `ESR_EL1`.FnV is 1.

For all other exceptions taken to EL1, the FAR_EL1 is **UNKNOWN**.

If a memory fault that sets FAR_EL1 is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL1 is taken from an Exception level that is using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_EL1 are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store is **CONSTRAINED UNPREDICTABLE**. See ‘Out of range VA’ in Appendix K1 Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see ‘Address tagging in AArch64 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
Execution at EL0 makes FAR_EL1 become **UNKNOWN**.

---

**Note**

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

---

FAR_EL1 is made **UNKNOWN** on an exception return from EL1.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the FAR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic FAR_EL1 or FAR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, FAR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.FAR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x220];
  else
    return FAR_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return FAR_EL2;
  else
    return FAR_EL1;
elsif PSTATE.EL == EL3 then
  return FAR_EL1;

**MSR FAR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x220] = X[t];
    else
        FAR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            FAR_EL2 = X[t];
        else
            UNDEFINED;
        elsif PSTATE.EL == EL3 then
            FAR_EL1 = X[t];
    end if
endif

MRS <Xt>, FAR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x220];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if EL2Enabled() && HCR_EL2.E2H == '1' then
            return FAR_EL1;
        else
            UNDEFINED;
        elsif PSTATE.EL == EL3 then
            if EL2Enabled() && HCR_EL2.E2H == '1' then
                return FAR_EL1;
            else
                UNDEFINED;
            end if
    end if
endif

MSR FAR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '101' then
        NVMem[0x220] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        FAR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        FAR_EL1 = X[t];
    else
        UNDEFINED;
MRS <Xt>, FAR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return FAR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return FAR_EL2;
else PSTATE.EL == EL3 then
    return FAR_EL2;

MSR FAR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        FAR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t];
else PSTATE.EL == EL3 then
    FAR_EL2 = X[t];
FAR_EL2, Fault Address Register (EL2)

The FAR_EL2 characteristics are:

**Purpose**

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL2.

**Configuration**

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HDFAR[31:0].

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HIFAR[31:0].

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register DFAR[31:0] when HaveEL(EL2).

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register IFAR[31:0] when HaveEL(EL2).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

FAR_EL2 is a 64-bit register.

**Field descriptions**

The FAR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Faulting Virtual Address for synchronous exceptions taken to EL2 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Faulting Virtual Address for synchronous exceptions taken to EL2. Exceptions that set the FAR_EL2 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), PC alignment faults (EC 0x22), and Watchpoints (EC 0x34 or 0x35). ESR_EL2.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL2 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL2.FnV is 0, and the FAR_EL2 is UNKNOWN if ESR_EL2.FnV is 1.

For all other exceptions taken to EL2, the FAR_EL2 is UNKNOWN.

If a memory fault that sets FAR_EL2 is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL2 is taken from an Exception level that is using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:
• The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE. See 'Out of range VA' in Appendix K1 Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

• The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution at EL1 or EL0 makes FAR_EL2 become UNKNOWN.

---

**Note**

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

---

FAR_EL2 is made UNKNOWN on an exception return from EL2.

This field resets to an architecturally UNKNOWN value.

### Accessing the FAR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic FAR_EL2 or FAR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, FAR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return FAR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        return FAR_EL2;
    elsif PSTATE.EL == EL3 then
        return FAR_EL2;
else
    UNDEFINED;

**MSR FAR_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        FAR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t];

MRS <Xt>, FAR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x220];
    else
        return FAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return FAR_EL2;
    else
        FAR_EL1 = X[t];
else
    return FAR_EL1;
elsif PSTATE.EL == EL3 then
    return FAR_EL1;

MSR FAR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x220];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        FAR_EL2 = X[t];
    else
        FAR_EL1 = X[t];
else
    FAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL1 = X[t];
FAR_EL3, Fault Address Register (EL3)

The FAR_EL3 characteristics are:

**Purpose**

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort and PC alignment fault exceptions that are taken to EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to FAR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

FAR_EL3 is a 64-bit register.

**Field descriptions**

The FAR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Faulting Virtual Address for synchronous exceptions taken to EL3

Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL3. Exceptions that set the FAR_EL3 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), and PC alignment faults (EC 0x22). ESR_EL3.EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which TCR_ELx.TBI{<0|1>} == 1 for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL3 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if ESR_EL3.FnV is 0, and the FAR_EL3 is UNKNOWN if ESR_EL3.FnV is 1.

For all other exceptions taken to EL3, the FAR_EL3 is UNKNOWN.

If a memory fault that sets FAR_EL3 is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

If the exception that updates FAR_EL3 is taken from an Exception Level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE. See 'Out of range VA' in Appendix K1 Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Execution at EL2, EL1 or EL0 makes FAR_EL3 become UNKNOWN.
Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or data abort. It is the lowest address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL3 is made **UNKNOWN** on an exception return from EL3.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the FAR_EL3**

Accesses to this register use the following encodings:

**MRS <Xt>, FAR_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return FAR_EL3;
```

**MSR FAR_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    FAR_EL3 = X[t];
```
FPCR, Floating-point Control Register

The FPCR characteristics are:

**Purpose**

Controls floating-point behavior.

**Configuration**

The named fields in this register map to the equivalent fields in the AArch32 FPSCR.

It is **IMPLEMENTATION DEFINED** whether the Len and Stride fields can be programmed to non-zero values, which will cause some AArch32 floating-point instruction encodings to be **UNDEFINED**, or whether these fields are **RAZ**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

FPCR is a 64-bit register.

**Field descriptions**

The FPCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>AHP</td>
<td>Alternative half-precision control bit:</td>
</tr>
<tr>
<td>61</td>
<td>DN</td>
<td>Default NaN mode control bit:</td>
</tr>
<tr>
<td>60</td>
<td>FZ</td>
<td>IEEE half-precision format selected.</td>
</tr>
<tr>
<td>59</td>
<td>RMode</td>
<td>Alternative half-precision format selected.</td>
</tr>
<tr>
<td>58</td>
<td>FZ16</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
<tr>
<td>57</td>
<td>Len</td>
<td>Any operation involving one or more NaNs returns the Default NaN.</td>
</tr>
<tr>
<td>56</td>
<td>IDE</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
<tr>
<td>55</td>
<td>UFE</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
<tr>
<td>54</td>
<td>OFE</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
<tr>
<td>53</td>
<td>DZE</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
<tr>
<td>52</td>
<td>IOE</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
</tbody>
</table>

This bit is only used for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the ARMv8.2-FP16 extension always use the IEEE half-precision format, and ignore the value of this bit.

This field resets to an architecturally **UNKNOWN** value.
The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

This field resets to an architecturally *UNKNOWN* value.

**FZ, bit [24]**

Flush-to-zero mode control bit:

<table>
<thead>
<tr>
<th>FZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Flush-to-zero mode disabled. Behavior of the floating-point system</td>
</tr>
<tr>
<td></td>
<td>is fully compliant with the IEEE 754 standard.</td>
</tr>
<tr>
<td>0b1</td>
<td>Flush-to-zero mode enabled.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

This bit has no effect on half-precision calculations.

This field resets to an architecturally *UNKNOWN* value.

**RMode, bits [23:22]**

Rounding Mode control field. The encoding of this field is:

<table>
<thead>
<tr>
<th>RMode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Round to Nearest (RN) mode.</td>
</tr>
<tr>
<td>0b01</td>
<td>Round towards Plus Infinity (RP) mode.</td>
</tr>
<tr>
<td>0b10</td>
<td>Round towards Minus Infinity (RM) mode.</td>
</tr>
<tr>
<td>0b11</td>
<td>Round towards Zero (RZ) mode.</td>
</tr>
</tbody>
</table>

The specified rounding mode is used by both scalar and Advanced SIMD floating-point instructions.

This field resets to an architecturally *UNKNOWN* value.

**Stride, bits [21:20]**

This field has no function in AArch64 state, and non-zero values are ignored during execution in AArch64 state. It is included only for context saving and restoration of the AArch32 FPSCR.Stride field.

This field resets to an architecturally *UNKNOWN* value.

**FZ16, bit [19]**

*When ARMv8.2-FP16 is implemented:*

Flush-to-zero mode control bit on half-precision data-processing instructions:

<table>
<thead>
<tr>
<th>FZ16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Flush-to-zero mode disabled. Behavior of the floating-point system</td>
</tr>
<tr>
<td></td>
<td>system is fully compliant with the IEEE 754 standard.</td>
</tr>
<tr>
<td>0b1</td>
<td>Flush-to-zero mode enabled.</td>
</tr>
</tbody>
</table>

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations. A half-precision floating-point number that is flushed to zero as a result of the value of the FZ16 bit does not generate an Input Denormal exception.

This field resets to an architecturally *UNKNOWN* value.

*Otherwise:*

Reserved, RES0.
Len, bits [18:16]

This field has no function in AArch64 state, and non-zero values are ignored during execution in AArch64 state. It is included only for context saving and restoration of the AArch32 FPSCR.Len field.

This field resets to an architecturally \textit{UNKNOWN} value.

IDE, bit [15]

Input Denormal floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>IDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untracked exception handling selected. If the floating-point exception occurs then the FPSR.IDC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IDC bit. The trap handling software can decide whether to set the FPSR.IDC bit to 1.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

This field resets to an architecturally \textit{UNKNOWN} value.

Bits [14:13]

Reserved, RES0.

IXE, bit [12]

Inexact floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>IXE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untracked exception handling selected. If the floating-point exception occurs then the FPSR.IXC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracked exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IXC bit. The trap handling software can decide whether to set the FPSR.IXC bit to 1.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

This field resets to an architecturally \textit{UNKNOWN} value.

UFE, bit [11]

Underflow floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>UFE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untracked exception handling selected. If the floating-point exception occurs then the FPSR.UFC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracked exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.UFC bit. The trap handling software can decide whether to set the FPSR.UFC bit to 1.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

This field resets to an architecturally \textit{UNKNOWN} value.
**OFE, bit [10]**

Overflow floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>OFE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the FPSR.OFC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.OFC bit. The trap handling software can decide whether to set the FPSR.OFC bit to 1.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

**DZE, bit [9]**

Divide by Zero floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>DZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the FPSR.DZC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.DZC bit. The trap handling software can decide whether to set the FPSR.DZC bit to 1.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

**IOE, bit [8]**

Invalid Operation floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>IOE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the FPSR.IOC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IOC bit. The trap handling software can decide whether to set the FPSR.IOC bit to 1.</td>
</tr>
</tbody>
</table>

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

**Bits [7:0]**

Reserved, RES0.

**Accessing the FPCR**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x00);
    else
      AArch64.SystemAccessTrap(EL1, 0x07);
    endif
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
    AArch64.SystemAccessTrap(EL3, 0x07);
  else
    return FPCR;
  endif
else
  MSR FPCR, <Xt>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CPACR_EL1.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x00);
    else
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CPACR_EL1.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t];
    elsif PSTATE.EL == EL1 then
        if CPACR_EL1.FPEN == 'x0' then
            AArch64.SystemAccessTrap(EL1, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x07);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            AArch64.SystemAccessTrap(EL3, 0x07);
        else
            FPCR = X[t];
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
                AArch64.SystemAccessTrap(EL2, 0x07);
            elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
                AArch64.SystemAccessTrap(EL2, 0x07);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
                AArch64.SystemAccessTrap(EL3, 0x07);
            else
                FPCR = X[t];
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TFP == '1' then
                AArch64.SystemAccessTrap(EL3, 0x07);
            else
                FPCR = X[t];
    else
        FPCR = X[t];

27/09/2019 18:22; 6134483bd414dc8c12a90c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**FPEXC32_EL2, Floating-Point Exception Control register**

The FPEXC32_EL2 characteristics are:

**Purpose**

Allows access to the AArch32 register FPEXC from AArch64 state only. Its value has no effect on execution in AArch64 state.

**Configuration**

AArch64 System register FPEXC32_EL2 bits [31:0] are architecturally mapped to AArch32 System register FPEXC[31:0].

If EL1 cannot use AArch32, this register is UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

FPEXC32_EL2 is a 64-bit register.

**Field descriptions**

The FPEXC32_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>EX, bit [31]</td>
</tr>
<tr>
<td>30</td>
<td>EN, bit [30]</td>
</tr>
</tbody>
</table>

**EX, bit [31]**

Exception bit. From Armv8, this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

**EN, bit [30]**

Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:

- VMSR accesses to the FPEXC or FPSID.
- VMRS accesses from the FPEXC, FPSID, MVFR0, MVFR1, or MVFR2.
Meaning

<table>
<thead>
<tr>
<th>EN</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to the FPSCR, and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers, are UNDEFINED at all Exception levels.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control permits access to the Advanced SIMD and floating-point functionality at all Exception levels.</td>
</tr>
</tbody>
</table>

Execution of floating-point and Advanced SIMD instructions in AArch32 state can be disabled or trapped by the following controls:

- **CPACR**.cp10, or, if executing at EL0, **CPACR_EL1**. FPEN.
- **FPEXC.EN**.
- If executing in Non-secure state:
  - **HCPtr.TCP10**, or if EL2 is using AArch64, **CPTR_EL2.TFP**.
  - **NSACR**.cp10, or if EL3 is using AArch64, **CPTR_EL3.TFP**.
- For Advanced SIMD instructions only:
  - **CPACR.ASEDIS**.
  - If executing in Non-secure state, **HCPtr.TASE** and **NSACR.NSTRCDIS**.

See the descriptions of the controls for more information.

**Note**

When executing at EL0 using AArch32:

- If EL1 is using AArch64 then behavior is as if the value of FPEXC.EN is 1.
- If EL2 is using AArch64 and enabled in the current Security state, and the value of **HCR_EL2**. {RW, TGE} is {1, 1} then behavior is as if the value of FPEXC.EN is 1.
- If EL2 is using AArch64 and enabled in the current Security state, and the value of **HCR_EL2**. {RW, TGE} is {0, 1} then it is IMPLEMENTATION DEFINED whether the behavior is:
  - As if the value of FPEXC.EN is 1.
  - Determined by the value of FPEXC32_EL2.EN, as described in this field description. However, Arm deprecates using the value of FPEXC32_EL2.EN to determine behavior.

This field resets to an architecturally **UNKNOWN** value.

**DEX, bit [29]**

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC32_EL2.TFV field is valid.

The meaning of this bit is:

<table>
<thead>
<tr>
<th>DEX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC32_EL2.TFV is RW then it is invalid and <strong>UNKNOWN</strong>. If FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and <strong>UNKNOWN</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>The exception was generated during the execution of an unallocated encoding. FPEXC32_EL2.TFV is valid and indicates the cause of the exception.</td>
</tr>
</tbody>
</table>

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the AArch32 FPSCR. {Stride, Len} fields as RAZ, this bit is **RES0**.

This field resets to an architecturally **UNKNOWN** value.
FP2V, bit [28]

FPINST2 instruction valid bit. From Armv8, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

VV, bit [27]

VECITR valid bit. From Armv8, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC.DEX is 1. When valid, it indicates the cause of the exception and therefore whether the FPEXC.{IDF, IXF, OFF, DZF, IOF} bits are valid.

<table>
<thead>
<tr>
<th>TFV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VSQRT, or VSUB instruction when one or both of FPSCR. {Stride, Len} was non-zero. If the FPEXC.{IDF, IXF, OFF, DZF, IOF} bits are RW then they are invalid and UNKNOWN.</td>
</tr>
<tr>
<td>0b1</td>
<td>FPEXC.{IDF, IXF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception.</td>
</tr>
</tbody>
</table>

This bit returns a status value and ignores writes.

When the value of FPEXC.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

On an implementation that supports the trapping of floating-point exceptions and implements FPSCR. {Stride, Len} as RAZ, this bit is RAO/WI.

This field resets to an architecturally UNKNOWN value.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector iteration count. From Armv8, this field is RES1.

This field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while FPSCR. FZ was 1:

<table>
<thead>
<tr>
<th>IDF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input Denormal exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input Denormal exception has occurred.</td>
</tr>
</tbody>
</table>

Input Denormal exceptions can occur only when FPSCR. FZ is 1.

Note
A half-precision floating-point value that is flushed to zero because the value of \texttt{FPSCR.FZ16} is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and \texttt{UNKNOWN}.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally \texttt{UNKNOWN} value.

\textbf{Bits [6:5]}

Reserved, \texttt{RES0}.

\textbf{IXF, bit [4]}

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while \texttt{FPSCR.IXE} was 1:

<table>
<thead>
<tr>
<th>IXF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Inexact exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inexact exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and \texttt{UNKNOWN}.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally \texttt{UNKNOWN} value.

\textbf{UFF, bit [3]}

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while \texttt{FPSCR.UFE} was 1:

<table>
<thead>
<tr>
<th>UFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow exception has occurred.</td>
</tr>
</tbody>
</table>

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when \texttt{FPSCR.FZ16} is 0.
- Otherwise only when \texttt{FPSCR.FZ} is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and \texttt{UNKNOWN}.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally \texttt{UNKNOWN} value.

\textbf{OFF, bit [2]}

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while \texttt{FPSCR.OFE} was 1:

<table>
<thead>
<tr>
<th>OFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.
When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

### DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while **FPSCR.DZE** was 1:

<table>
<thead>
<tr>
<th>DZF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Divide by Zero exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Divide by Zero exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

### IOF, bit [0]

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while **FPSCR.IOE** was 1:

<table>
<thead>
<tr>
<th>IOF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Invalid Operation exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invalid Operation exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the FPEXC32_EL2

Accesses to this register use the following encodings:

**MRS <Xt>, FPEXC32_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x07);
   elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x07);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
   else
      FPEXC32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
   if CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
   else
      FPEXC32_EL2 = X[t];

MSR FPEXC32_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x07);
   elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x07);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
   else
      FPEXC32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
   if CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
   else
      FPEXC32_EL2 = X[t];
FPSR, Floating-point Status Register

The FPSR characteristics are:

**Purpose**

Provides floating-point system status information.

**Configuration**

The named fields in this register map to the equivalent fields in the AArch32 FPSR.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

FPSR is a 64-bit register.

**Field descriptions**

The FPSR bit assignments are:

|   | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| N | Z  | C  | V  | QC | RES0| DC | RES0| IXC | UFC | OFC | DZC | IOC | RES0| RES0| RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 |
|---|----|----|----|----|-----|----|-----|-----|-----|-----|-----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

**Bits [63:32]**

Reserved, **RES0**.

**N, bit [31]**

Negative condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.N flag instead.

This field resets to an architecturally **UNKNOWN** value.

**Z, bit [30]**

Zero condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.Z flag instead.

This field resets to an architecturally **UNKNOWN** value.

**C, bit [29]**

Carry condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.C flag instead.

This field resets to an architecturally **UNKNOWN** value.

**V, bit [28]**

Overflow condition flag for AArch32 floating-point comparison operations. AArch64 floating-point comparisons set the PSTATE.V flag instead.
This field resets to an architecturally **UNKNOWN** value.

**QC, bit [27]**

Cumulative saturation bit, Advanced SIMD only. This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated since 0 was last written to this bit.

This field resets to an architecturally **UNKNOWN** value.

**Bits [26:8]**

Reserved, RES0.

**IDC, bit [7]**

Input Denormal cumulative floating-point exception bit. This bit is set to 1 to indicate that the Input Denormal floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.IDE bit. This bit is only set to 1 to indicate a floating-point exception if FPCR.IDE is 0, or if trapping software sets it.

This field resets to an architecturally **UNKNOWN** value.

**Bits [6:5]**

Reserved, RES0.

**IXC, bit [4]**

Inexact cumulative floating-point exception bit. This bit is set to 1 to indicate that the Inexact exception floating-point has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.IXE bit. This bit is only set to 1 to indicate a floating-point exception if FPCR.IXE is 0, or if trapping software sets it.

This field resets to an architecturally **UNKNOWN** value.

**UFC, bit [3]**

Underflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Underflow floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.UFE bit. This bit is only set to 1 to indicate a floating-point exception if FPCR.UFE is 0, or if trapping software sets it.

This field resets to an architecturally **UNKNOWN** value.

**OFC, bit [2]**

Overflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Overflow floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.OFE bit. This bit is only set to 1 to indicate a floating-point exception if FPCR.OFE is 0, or if trapping software sets it.

This field resets to an architecturally **UNKNOWN** value.

**DZC, bit [1]**

Divide by Zero cumulative floating-point exception bit. This bit is set to 1 to indicate that the Divide by Zero floating-point exception has occurred since 0 was last written to this bit.
How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.DZE bit. This bit is only set to 1 to indicate a floating-point exception if FPCR.DZE is 0, or if trapping software sets it.

This field resets to an architecturally UNKNOWN value.

**IOC, bit [0]**

Invalid Operation cumulative floating-point exception bit. This bit is set to 1 to indicate that the Invalid Operation floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the FPCR.IOE bit. This bit is only set to 1 to indicate a floating-point exception if FPCR.IOE is 0, or if trapping software sets it.

This field resets to an architecturally UNKNOWN value.

**Accessing the FPSR**

Accesses to this register use the following encodings:

\[
\text{MRS} \ <Xt>, \ FPSR
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CPACR_EL1.FPEN != '11' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x00);
    else
      AArch64.SystemAccessTrap(EL1, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
    AArch64.SystemAccessTrap(EL3, 0x07);
  else
    return FPSR;
  endif
else
  if PSTATE.EL == EL1 then
    if CPACR_EL1.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.TFP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
    else
      return FPSR;
    endif
  elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
    else
      return FPSR;
    endif
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
    else
      return FPSR;
    endif
  endif
endif

MSR FPSR, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !ELEnabled() && HCR_EL2.<E2H,TGE> == '11' && CPACR_EL1.FPEN != '11' then
    AArch64.SystemAccessTrap(EL2, 0x00);
  else
    AArch64.SystemAccessTrap(EL1, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CPACR_EL1.FPEN != '11' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x07);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
    AArch64.SystemAccessTrap(EL3, 0x07);
  else
    FPSR = X[t];
  endif;
else
  if PSTATE.EL == EL1 then
    if CPACR_EL1.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
      AArch64.SystemAccessTrap(EL3, 0x07);
    else
      FPSR = X[t];
    endif;
  else
    if PSTATE.EL == EL2 then
      if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
      elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
      else
        FPSR = X[t];
      endif;
    else
      if PSTATE.EL == EL3 then
        if CPTR_EL3.TFP == '1' then
          AArch64.SystemAccessTrap(EL3, 0x07);
        else
          FPSR = X[t];
        endif;
      endif;
    endif;
  endif;
endif;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbf63431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GCR_EL1, Tag Control Register.

The GCR_EL1 characteristics are:

**Purpose**

Tag Control Register.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to GCR_EL1 are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

GCR_EL1 is a 64-bit register.

**Field descriptions**

The GCR_EL1 bit assignments are:

```
  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32
<p>| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RRND</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exclude</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Bits [63:17]**

Reserved, RES0.

**RRND, bit [16]**

Controls whether RandomTag() generates a deterministic value solely based on the contents of RGSR_EL1, or a non-deterministic value.

<table>
<thead>
<tr>
<th>RRND</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Generate a deterministic value based on RGSR_EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Generate an IMPLEMENTATION DEFINED non-deterministic value.</td>
</tr>
</tbody>
</table>

**Note**

When the value of GCR_EL1.RRND is 1, the value generated does not need to be cryptographically random.

A similar algorithm to that used when RRND=0 but with free running clock is sufficient.

This field resets to an architecturally **UNKNOWN** value.

**Exclude, bits [15:0]**

Allocation Tag values excluded from selection by ChooseNonExcludedTag().

This field resets to an architecturally **UNKNOWN** value.
Accessing the GCR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, GCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsf EL3Enabled() && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return GCR_EL1;
elsf PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    GCR_EL1 = X[t];
elsf PSTATE.EL == EL3 then
  return GCR_EL1;

MSR GCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsf EL3Enabled() && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    GCR_EL1 = X[t];
elsf PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    GCR_EL1 = X[t];
elsf PSTATE.EL == EL3 then
  GCR_EL1 = X[t];
The GMID_EL1 characteristics are:

**Purpose**

Indicates the block size that is accessed by the LDGM and STGM System instructions.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to GMID_EL1 are **UNDEFINED**.

**Attributes**

GMID_EL1 is a 64-bit register.

**Field descriptions**

The GMID_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>BS, bits [3:0]</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**BS, bits [3:0]**

Log2 of the block size in words. The minimum supported size is 16B (value == 2) and the maximum is 256B (value == 6).

**Accessing the GMID_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, GMID_EL1
```

<table>
<thead>
<tr>
<th>CRn</th>
<th>op0</th>
<th>op1</th>
<th>op2</th>
<th>CRm</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>0b11</td>
<td>0b001</td>
<td>0b100</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if;
    else
        UNDEFINED;
    end if;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID5 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return GMID_EL1;
    end if;
elsif PSTATE.EL == EL2 then
    return GMID_EL1;
elsif PSTATE.EL == EL3 then
    return GMID_EL1;
end if;
HACR_EL2, Hypervisor Auxiliary Control Register

The HACR_EL2 characteristics are:

**Purpose**

Controls trapping to EL2 of IMPLEMENTATION DEFINED aspects of EL1 or EL0 operation.

**Note**

Arm recommends that the values in this register do not cause unnecessary traps to EL2 when HCR_EL2.E2H, TGE == {1, 1}.

**Configuration**

AArch64 System register HACR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HACR[31:0].

If EL2 is not implemented, this register is RES from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HACR_EL2 is a 64-bit register.

**Field descriptions**

The HACR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:0]**

This field resets to an architecturally UNKNOWN value.

**Accessing the HACR_EL2**

Accesses to this register use the following encodings:

```
MRS <Xt>, HACR_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elself PSTATE.EL == EL2 then
    return HACR_EL2;
elself PSTATE.EL == EL3 then
    return HACR_EL2;

MSR HACR_EL2, <Xt>
The HAFGRTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MRS reads of Activity Monitors System registers.

**Configuration**

This register is present only when AMUv1 is implemented and ARMv8.6-FGT is implemented. Otherwise, direct accesses to HAFGRTR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HAFGRTR_EL2 is a 64-bit register.

**Field descriptions**

The HAFGRTR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>AMEVTYPE14_EL0</td>
</tr>
<tr>
<td>61</td>
<td>AMEVCNTR14_EL0</td>
</tr>
<tr>
<td>60</td>
<td>AMEVTYPE13_EL0</td>
</tr>
<tr>
<td>59</td>
<td>AMEVCNTR13_EL0</td>
</tr>
<tr>
<td>58</td>
<td>AMEVTYPE12_EL0</td>
</tr>
<tr>
<td>57</td>
<td>AMEVCNTR12_EL0</td>
</tr>
<tr>
<td>56</td>
<td>AMEVTYPE11_EL0</td>
</tr>
<tr>
<td>55</td>
<td>AMEVCNTR11_EL0</td>
</tr>
<tr>
<td>54</td>
<td>AMEVTYPE10_EL0</td>
</tr>
<tr>
<td>53</td>
<td>AMEVCNTR10_EL0</td>
</tr>
<tr>
<td>52</td>
<td>AMEVTYPE9_EL0</td>
</tr>
<tr>
<td>51</td>
<td>AMEVCNTR9_EL0</td>
</tr>
<tr>
<td>50</td>
<td>AMEVTYPE8_EL0</td>
</tr>
<tr>
<td>49</td>
<td>AMEVCNTR8_EL0</td>
</tr>
<tr>
<td>48</td>
<td>AMEVTYPE7_EL0</td>
</tr>
<tr>
<td>47</td>
<td>AMEVCNTR7_EL0</td>
</tr>
<tr>
<td>46</td>
<td>AMEVTYPE6_EL0</td>
</tr>
<tr>
<td>45</td>
<td>AMEVCNTR6_EL0</td>
</tr>
<tr>
<td>44</td>
<td>AMEVTYPE5_EL0</td>
</tr>
<tr>
<td>43</td>
<td>AMEVCNTR5_EL0</td>
</tr>
<tr>
<td>42</td>
<td>AMEVTYPE4_EL0</td>
</tr>
<tr>
<td>41</td>
<td>AMEVCNTR4_EL0</td>
</tr>
<tr>
<td>40</td>
<td>AMEVTYPE3_EL0</td>
</tr>
<tr>
<td>39</td>
<td>AMEVCNTR3_EL0</td>
</tr>
<tr>
<td>38</td>
<td>AMEVTYPE2_EL0</td>
</tr>
<tr>
<td>37</td>
<td>AMEVCNTR2_EL0</td>
</tr>
<tr>
<td>36</td>
<td>AMEVTYPE1_EL0</td>
</tr>
<tr>
<td>35</td>
<td>AMEVCNTR1_EL0</td>
</tr>
<tr>
<td>34</td>
<td>AMEVTYPE0_EL0</td>
</tr>
<tr>
<td>33</td>
<td>AMEVCNTR0_EL0</td>
</tr>
</tbody>
</table>

**Bits [63:50]**

Reserved, RES0.

**AMEVTYPER115_EL0, bit [49]**

Trap MRS reads of AMEVTYPE1<15>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPE1<15> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPE115_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPE1&lt;15&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPE1&lt;15&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.(E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVTYPE1&lt;15&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVTYPE1&lt;15&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x63.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR115_EL0, bit [48]**

Trap MRS reads of AMEVCNTR1[15>] at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1[15>] at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR115_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1[15&gt;] at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1[15&gt;] at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.[E2H,TGE] != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTe == 1, then, unless the read generates a higher priority exception:  
• MRS reads of AMEVCNTR1[15>] at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
• MRC reads of AMEVCNTR1[15>] at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER114_EL0, bit [47]**

Trap MRS reads of AMEVTYPER1[14>] at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1[14>] at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER114_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1[14&gt;] at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1[14&gt;] at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1              | If EL2 is implemented and enabled in the current Security state, HCR_EL2.[E2H,TGE] != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTe == 1, then, unless the read generates a higher priority exception:  
• MRS reads of AMEVTYPER1[14>] at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
• MRC reads of AMEVTYPER1[14>] at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR114_EL0, bit [46]**

Trap MRS reads of AMEVCNTR1[14>] at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1[14>] at EL0 using AArch32 when EL1 is using AArch64 to EL2.
### AMEVCNTR114_EL0, bit [45]

| 0b0 | MRS reads of AMEVCNTR1<14>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<14> at EL0 using AArch32 are not affected by this bit. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:
  • MRS reads of AMEVCNTR1<14>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  • MRC reads of AMEVCNTR1<14> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

### AMEVTYPER113_EL0, bit [45]

Trap MRS reads of AMEVTYPER1<13>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<13> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| 0b0 | MRS reads of AMEVTYPER1<13>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<13> at EL0 using AArch32 are not affected by this bit. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:
  • MRS reads of AMEVTYPER1<13>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  • MRC reads of AMEVTYPER1<13> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

### AMEVCNTR113_EL0, bit [44]

Trap MRS reads of AMEVCNTR1<13>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<13> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
AMEVCNTR113_EL0

| Meaning | 0b0 | MRS reads of AMEVCNTR1<13>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<13> at EL0 using AArch32 are not affected by this bit. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, \( HCR\_EL2\).{E2H,TGE} \(!=\) \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or \( SCR\_EL3\).FGTEn \(!=\) 1, then, unless the read generates a higher priority exception: |
| | • MRS reads of AMEVCNTR1<13>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. |
| | • MRC reads of AMEVCNTR1<13> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

AMEVTYPER112_EL0, bit [43]

Trap MRS reads of AMEVTYPER1<12>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<12> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| Meaning | 0b0 | MRS reads of AMEVTYPER1<12>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<12> at EL0 using AArch32 are not affected by this bit. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, \( HCR\_EL2\).{E2H,TGE} \(!=\) \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or \( SCR\_EL3\).FGTEn \(!=\) 1, then, unless the read generates a higher priority exception: |
| | • MRS reads of AMEVTYPER1<12>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. |
| | • MRC reads of AMEVTYPER1<12> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

AMEVCNTR112_EL0, bit [42]

Trap MRS reads of AMEVCNTR1<12>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<12> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
AMEVCNTR112_EL0

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;12&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;12&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVCNTR1&lt;12&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVCNTR1&lt;12&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

AMEVTYPER111_EL0, bit [41]

Trap MRS reads of AMEVTYPER1<11>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<11> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;11&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;11&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVTYPER1&lt;11&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVTYPER1&lt;11&gt;_EL0 at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

AMEVCNTR111_EL0, bit [40]

Trap MRS reads of AMEVCNTR1<11>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<11> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
AMEVCNTR111_EL0

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> MRS reads of AMEVCNTR1&lt;11&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;11&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td><strong>0b1</strong> If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td>• MRS reads of AMEVCNTR1&lt;11&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td>• MRC reads of AMEVCNTR1&lt;11&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

AMEVTYPER110_EL0, bit [39]

Trap MRS reads of AMEVTYPER1<10>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<10> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

AMEVTYPER110_EL0

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> MRS reads of AMEVTYPER1&lt;10&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;10&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td><strong>0b1</strong> If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td>• MRS reads of AMEVTYPER1&lt;10&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td>• MRC reads of AMEVTYPER1&lt;10&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

AMEVCNTR110_EL0, bit [38]

Trap MRS reads of AMEVCNTR1<10>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<10> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
<table>
<thead>
<tr>
<th>AMEVCNTR110_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;10&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;10&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
|                  | • MRS reads of AMEVCNTR1<10>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
|                  | • MRC reads of AMEVCNTR1<10> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER19_EL0, bit [37]**

Trap MRS reads of AMEVTYPER1<9>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<9> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER19_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;9&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;9&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
|                  | • MRS reads of AMEVTYPER1<9>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
|                  | • MRC reads of AMEVTYPER1<9> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR19_EL0, bit [36]**

Trap MRS reads of AMEVCNTR1<9>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<9> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR19_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;9&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;9&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
|                  | • MRS reads of AMEVCNTR1<9>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
|                  | • MRC reads of AMEVCNTR1<9> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |
In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER18_EL0, bit [35]**

Trap MRS reads of AMEVTYPER1<8> EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<8> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER18_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;8&gt; EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;8&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
• MRS reads of AMEVTYPER1<8> EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
• MRC reads of AMEVTYPER1<8> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR18_EL0, bit [34]**

Trap MRS reads of AMEVCNTR1<8> EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<8> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR18_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;8&gt; EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;8&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1            | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
• MRS reads of AMEVCNTR1<8> EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
• MRC reads of AMEVCNTR1<8> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER17_EL0, bit [33]**

Trap MRS reads of AMEVTYPER1<7> EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<7> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
### AMEVTYPER17_EL0

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;7&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;7&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  \* MRS reads of AMEVTYPER1<7>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  \* MRC reads of AMEVTYPER1<7> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

### AMEVCNTR17_EL0, bit [32]

Trap MRS reads of AMEVCNTR1<7>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<7> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;7&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;7&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  \* MRS reads of AMEVCNTR1<7>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  \* MRC reads of AMEVCNTR1<7> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

### AMEVTYPER16_EL0, bit [31]

Trap MRS reads of AMEVTYPER1<6>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<6> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;6&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;6&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  \* MRS reads of AMEVTYPER1<6>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  \* MRC reads of AMEVTYPER1<6> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |
In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR16_EL0, bit [30]**

Trap MRS reads of AMEVCNTR1<6> at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<6> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR16_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;6&gt; at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;6&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, ( \text{HCR_EL2}.{E2H,TGE} \neq {1,1}, ) EL1 is using AArch64, and either EL3 is not implemented or ( \text{SCR_EL3}.\text{FGTEn} == 1, ) then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVCNTR1&lt;6&gt; at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVCNTR1&lt;6&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER15_EL0, bit [29]**

Trap MRS reads of AMEVTYPER1<5> at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<5> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER15_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;5&gt; at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;5&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, ( \text{HCR_EL2}.{E2H,TGE} \neq {1,1}, ) EL1 is using AArch64, and either EL3 is not implemented or ( \text{SCR_EL3}.\text{FGTEn} == 1, ) then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVTYPER1&lt;5&gt; at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVTYPER1&lt;5&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR15_EL0, bit [28]**

Trap MRS reads of AMEVCNTR1<5> at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<5> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
AMEVCNTR15_EL0, bit [26]

Trap MRS reads of AMEVCNTR1<5> at EL0 using AArch64 and MRC reads of AMEVCNTR1<5> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR15_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;5&gt; at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;5&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1            | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:

- MRS reads of AMEVCNTR1<5> at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
- MRC reads of AMEVCNTR1<5> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

AMEVTYPER14_EL0, bit [27]

Trap MRS reads of AMEVTYPER1<4> at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<4> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER14_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;4&gt; at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;4&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:

- MRS reads of AMEVTYPER1<4> at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
- MRC reads of AMEVTYPER1<4> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.
In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER13_EL0, bit [25]**

Trap MRS reads of AMEVTYPER1<3>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<3> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER13_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;3&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;3&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVTYPER1<3>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVTYPER1<3> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR13_EL0, bit [24]**

Trap MRS reads of AMEVCNTR1<3>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<3> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR13_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;3&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;3&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1            | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVCNTR1<3>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVCNTR1<3> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER12_EL0, bit [23]**

Trap MRS reads of AMEVTYPER1<2>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<2> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
AMEVTYPER12_EL0

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
</tbody>
</table>
| **0b1** | If EL2 is implemented and enabled in the current Security state, HCR_EL2 {E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVTYPER1<2>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVTYPER1<2> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

AMEVCNTR12_EL0, bit [22]

Trap MRS reads of AMEVCNTR1<2>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<2> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
</tbody>
</table>
| **0b1** | If EL2 is implemented and enabled in the current Security state, HCR_EL2 {E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVCNTR1<2>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVCNTR1<2> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

AMEVTYPER11_EL0, bit [21]

Trap MRS reads of AMEVTYPER1<1>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<1> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
</tbody>
</table>
| **0b1** | If EL2 is implemented and enabled in the current Security state, HCR_EL2 {E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVTYPER1<1>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVTYPER1<1> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |
In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR11_EL0, bit [20]**

Trap MRS reads of AMEVCNTR1<1>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<1> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR11_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;1&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;1&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1            | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  - MRS reads of AMEVCNTR1<1>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  - MRC reads of AMEVCNTR1<1> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVTYPER10_EL0, bit [19]**

Trap MRS reads of AMEVTYPER1<0>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<0> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVTYPER10_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVTYPER1&lt;0&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1&lt;0&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1             | If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  - MRS reads of AMEVTYPER1<0>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  - MRC reads of AMEVTYPER1<0> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR10_EL0, bit [18]**

Trap MRS reads of AMEVCNTR1<0>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<0> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
HAFGRTR_EL2, Hypervisor Activity Monitors Fine-Grained Read Trap Register

<table>
<thead>
<tr>
<th>AMEVCNTR10_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR1&lt;0&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1&lt;0&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1            | If EL2 is implemented and enabled in the current Security state, HCR_EL2.(E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVCNTR1<0>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVCNTR1<0> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

AMCNTEN1, bit [17]

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

• At EL1 and EL0 using AArch64: MRS reads of AMCNTENCLR1_EL0 and AMCNTENSET1_EL0.
• At EL0 using Arch32 when EL1 is using AArch64: MRC reads of AMCNTENCLR1 and AMCNTENSET1.

In a system where the PE resets into EL2, this field resets to 0.

Bits [16:5]

Reserved, RES0.

AMEVCNTR03_EL0, bit [4]

Trap MRS reads of AMEVCNTR0<3>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<3> at EL0 using AArch32 when EL1 is using AArch64 to EL2.
AMEVCNTR03_EL0, bit [3]

Trap MRS reads of AMEVCNTR0<3>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<3> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR03_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR0&lt;3&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0&lt;3&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.E2H,TGE != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVCNTR0&lt;3&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVCNTR0&lt;3&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

AMEVCNTR02_EL0, bit [2]

Trap MRS reads of AMEVCNTR0<2>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<2> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR02_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR0&lt;2&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0&lt;2&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.E2H,TGE != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVCNTR0&lt;2&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVCNTR0&lt;2&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

AMEVCNTR01_EL0, bit [1]

Trap MRS reads of AMEVCNTR0<1>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<1> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR01_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR0&lt;1&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0&lt;1&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.E2H,TGE != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of AMEVCNTR0&lt;1&gt;_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of AMEVCNTR0&lt;1&gt; at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

**AMEVCNTR0_EL0, bit [1]**

Trap MRS reads of AMEVCNTR0<0>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<0> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMEVCNTR0_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of AMEVCNTR0&lt;0&gt;_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0&lt;0&gt; at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1           | If EL2 is implemented and enabled in the current Security state, HCR_EL2.(E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads of AMEVCNTR0<0>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads of AMEVCNTR0<0> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**AMCNTEN0, bit [0]**

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

• At EL1 and EL0 using AArch64: MRS reads of AMCNTENCLR0_EL0 and AMCNTENSET0_EL0.
• At EL0 using Arch32 when EL1 is using AArch64: MRC reads of AMCNTENCLR0 and AMCNTENSET0.

<table>
<thead>
<tr>
<th>AMCNTEN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The operations listed above are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1      | If EL2 is implemented and enabled in the current Security state, HCR_EL2.(E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:  
  • MRS reads at EL1 and EL0 using AArch64 of AMCNTENCLR0_EL0 and AMCNTENSET0_EL0 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MRC reads at EL0 using AArch32 of AMCNTENCLR0 and AMCNTENSET0 are trapped to EL2 and reported with EC syndrome value 0x03, unless the read generates a higher priority exception. |

In a system where the PE resets into EL2, this field resets to 0.

**Accessing the HAFGRTR_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, HAFGRTR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

Page 632
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x1E8];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    HAFGRTR_EL2 = X[t];
  end if;
elsif PSTATE.EL == EL3 then
  return HAFGRTR_EL2;
end if;

MSR HAFGRTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x1E8] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    HAFGRTR_EL2 = X[t];
  end if;
elsif PSTATE.EL == EL3 then
  HAFGRTR_EL2 = X[t];
end if;
HCR_EL2, Hypervisor Configuration Register

The HCR_EL2 characteristics are:

**Purpose**

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

**Configuration**

AArch64 System register HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HCR[31:0].

AArch64 System register HCR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HCR2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

The bits in this register behave as if they are 0 for all purposes other than direct reads of the register if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HCR_EL2 is a 64-bit register.

**Field descriptions**

The HCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>TWEDEL</td>
</tr>
<tr>
<td>62</td>
<td>TWEDEn</td>
</tr>
<tr>
<td>61</td>
<td>TID5</td>
</tr>
<tr>
<td>60</td>
<td>DCT</td>
</tr>
<tr>
<td>59</td>
<td>ATA</td>
</tr>
<tr>
<td>58</td>
<td>TTLBOS</td>
</tr>
<tr>
<td>57</td>
<td>TTLBIS</td>
</tr>
<tr>
<td>56</td>
<td>EnSCXT</td>
</tr>
<tr>
<td>55</td>
<td>TOCUAMVOFFEN</td>
</tr>
<tr>
<td>54</td>
<td>NTICAB</td>
</tr>
<tr>
<td>53</td>
<td>TID4</td>
</tr>
<tr>
<td>52</td>
<td>RES0</td>
</tr>
<tr>
<td>51</td>
<td>FiEN</td>
</tr>
<tr>
<td>50</td>
<td>FWB</td>
</tr>
<tr>
<td>49</td>
<td>NV2</td>
</tr>
<tr>
<td>48</td>
<td>AT</td>
</tr>
<tr>
<td>47</td>
<td>NV1</td>
</tr>
<tr>
<td>46</td>
<td>NV</td>
</tr>
<tr>
<td>45</td>
<td>API</td>
</tr>
<tr>
<td>44</td>
<td>APK</td>
</tr>
<tr>
<td>43</td>
<td>TME</td>
</tr>
<tr>
<td>42</td>
<td>MIOCNCE</td>
</tr>
<tr>
<td>41</td>
<td>TEA</td>
</tr>
<tr>
<td>40</td>
<td>TERR</td>
</tr>
<tr>
<td>39</td>
<td>TLOR</td>
</tr>
<tr>
<td>38</td>
<td>E2H</td>
</tr>
<tr>
<td>37</td>
<td>ID</td>
</tr>
<tr>
<td>36</td>
<td>CD</td>
</tr>
<tr>
<td>35</td>
<td>RW</td>
</tr>
<tr>
<td>34</td>
<td>TRVM</td>
</tr>
<tr>
<td>33</td>
<td>HD</td>
</tr>
<tr>
<td>32</td>
<td>TDZ</td>
</tr>
<tr>
<td>31</td>
<td>TGE</td>
</tr>
<tr>
<td>30</td>
<td>TVM</td>
</tr>
<tr>
<td>29</td>
<td>TTLB</td>
</tr>
<tr>
<td>28</td>
<td>TPU</td>
</tr>
<tr>
<td>27</td>
<td>TPCP</td>
</tr>
<tr>
<td>26</td>
<td>TSW</td>
</tr>
<tr>
<td>25</td>
<td>TACR</td>
</tr>
<tr>
<td>24</td>
<td>TIDCP</td>
</tr>
<tr>
<td>23</td>
<td>TSC</td>
</tr>
<tr>
<td>22</td>
<td>TID3</td>
</tr>
<tr>
<td>21</td>
<td>TID2</td>
</tr>
<tr>
<td>20</td>
<td>TID1</td>
</tr>
<tr>
<td>19</td>
<td>TID0</td>
</tr>
<tr>
<td>18</td>
<td>TACR</td>
</tr>
<tr>
<td>17</td>
<td>TIDCP</td>
</tr>
<tr>
<td>16</td>
<td>TSC</td>
</tr>
<tr>
<td>15</td>
<td>TID3</td>
</tr>
<tr>
<td>14</td>
<td>TID2</td>
</tr>
<tr>
<td>13</td>
<td>TID1</td>
</tr>
<tr>
<td>12</td>
<td>TID0</td>
</tr>
<tr>
<td>11</td>
<td>TACR</td>
</tr>
<tr>
<td>10</td>
<td>TIDCP</td>
</tr>
<tr>
<td>9</td>
<td>TSC</td>
</tr>
<tr>
<td>8</td>
<td>TID3</td>
</tr>
<tr>
<td>7</td>
<td>TID2</td>
</tr>
<tr>
<td>6</td>
<td>TID1</td>
</tr>
<tr>
<td>5</td>
<td>TID0</td>
</tr>
<tr>
<td>4</td>
<td>TACR</td>
</tr>
<tr>
<td>3</td>
<td>TIDCP</td>
</tr>
<tr>
<td>2</td>
<td>TSC</td>
</tr>
<tr>
<td>1</td>
<td>TID3</td>
</tr>
<tr>
<td>0</td>
<td>TID2</td>
</tr>
</tbody>
</table>

**TWEDEL, bits [63:60]**

When ARMv8.6-TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when HCR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by HCR_EL2.TWE as $2^{(TWEDEL + 8)}$ cycles.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**TWDEn, bit [59]**

When ARMv8.6-TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by HCR_EL2.TWE.

<table>
<thead>
<tr>
<th>TWDEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>The delay for taking a WFE trap is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>001</td>
<td>The delay for taking a WFE trap is at least the number of cycles defined in HCR_EL2.TWEDEL.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TID5, bit [58]**

**When ARMv8.5-MemTag is implemented:**

Trap ID group 5. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- **GMID_EL1.**

<table>
<thead>
<tr>
<th>TID5</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified EL1 and EL0 accesses to ID group 5 registers are</td>
</tr>
<tr>
<td></td>
<td>trapped to EL2.</td>
</tr>
</tbody>
</table>

When the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has an Effective value of 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**DCT, bit [57]**

**When ARMv8.5-MemTag is implemented:**

Default Cacheability Tagging. When **HCR_EL2.DC** is in effect, controls whether stage 1 translations are treated as Tagged or Untagged.

<table>
<thead>
<tr>
<th>DCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 translations are treated as Untagged.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 translations are treated as Tagged.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

** Otherwise:**

Reserved, RES0.

**ATA, bit [56]**

**When ARMv8.5-MemTag is implemented:**

Allocation Tag Access. When **SCR_EL3.ATA=1** and **HCR_EL2.(E2H,TGE) != {1,1}**, controls EL1 and EL0 access to Allocation Tags.

When access is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

MRS and MSR instructions at EL1 using GCR_EL1, RGSR_EL1, TFSR_EL1, TFSR_EL2, or TFSRE0_EL1 that are not UNDEFINED are trapped to EL2.

<table>
<thead>
<tr>
<th>ATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access is prevented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access is not prevented.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.
This field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**TTLBOS, bit [55]**

When ARMv8.2-EVT is implemented:

Trap TLB maintenance instructions that operate on the Outer Shareable domain. Traps execution of those TLB maintenance instructions at EL1 using AArch64 to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- TLBI VMALLE1OS
- TLBI VAE1OS
- TLBI ASIDE1OS
- TLBI VAAE1OS
- TLBI VALE1OS
- TLBI VALE1OS
- TLBI RVAE1OS
- TLBI RVAAE1OS
- TLBI RVFAE1OS
- TLBI VAALE1OS
- TLBI RVALE1OS

<table>
<thead>
<tr>
<th>TTLBOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Execution of the specified instructions are trapped to EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**TTLBIS, bit [54]**

When ARMv8.2-EVT is implemented:

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of those TLB maintenance instructions at EL1 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, TLBI RVAE1IS, TLBI RVAAE1IS, TLBI RVFAE1IS, and TLBI VAALE1IS.
- When EL1 is using AArch32, TLBIALIS, TLBIMVAIS, TLBIASIDIS, TLBIMVAIS, TLBIMVALIS, and TLBIMVAALIS.

<table>
<thead>
<tr>
<th>TTLBIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Execution of the specified instructions are trapped to EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**EnSCXT, bit [53]**

*When ARMv8.0-CSV2 is implemented:*

Enable Access to the **SCXTNUM_EL1** and **SCXTNUM_EL0** registers. The defined values are:

<table>
<thead>
<tr>
<th>EnSCXT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When (HCR_EL2.TGE==0 or HCR_EL2.E2H==0) and EL2 is enabled in the current Security state, EL1 and EL0 access to <strong>SCXTNUM_EL0</strong> and EL1 access to <strong>SCXTNUM_EL1</strong> is disabled by this mechanism, causing an exception to EL2, and the values of these registers to be treated as 0. When ((HCR_EL2.TGE==1 and HCR_EL2.E2H==1) and EL2 is enabled in the current Security state, EL0 access to <strong>SCXTNUM_EL0</strong> is disabled by this mechanism, causing an exception to EL2, and the value of this register to be treated as 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause accesses to <strong>SCXTNUM_EL0</strong> or <strong>SCXTNUM_EL1</strong> to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of **HCR_EL2.E2H, TGE** is {1,1}, this bit has no effect on execution at EL0.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TOCU, bit [52]**

*When ARMv8.2-EVT is implemented:*

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When **SCTLR_EL1.UCI** is 1, HCR_EL2.(TGE, E2H) is not {1, 1}, and EL0 is using AArch64, **IC IVAU, DC CVAU**.
- When EL1 is using AArch64, **IC IVAU, IC IALLU, DC CVAU**.
- When EL1 is using AArch32, **ICIMVAU, ICIALLU, DCCMVAU**.

**Note**

An exception generated because an instruction is **UNDEFINED** at EL0 is higher priority than this trap to EL2. In addition:

- **IC IALLUIS** and **IC IALLU** are always **UNDEFINED** at EL0 using AArch64.
- **ICIMVAU, ICIALLU, ICIALLUS**, and **DCCMVAU** are always **UNDEFINED** at EL0 using AArch32.

<table>
<thead>
<tr>
<th>TOCU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Execution of the specified instructions are trapped to EL2.</td>
</tr>
</tbody>
</table>

If the Point of Unification is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.
If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.E2H, TGE is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMVOFFEN, bit [51]

When ARMv8.6-AMU is implemented:

Activity Monitors Virtual Offsets Enable.

<table>
<thead>
<tr>
<th>AMVOFFEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtualization of the Activity Monitors is disabled. Indirect reads of the virtual offset registers are zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtualization of the Activity Monitors is enabled.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

TICAB, bit [50]

When ARMv8.2-EVT is implemented:

Trap ICIALUIS/IC IALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, IC IALLUIS.
- When EL1 is using AArch32, ICIALUIS.

<table>
<thead>
<tr>
<th>TICAB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 execution of the specified instructions is trapped to EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-EVT is not implemented, this field is RES0.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.E2H, TGE is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID4, bit [49]

When ARMv8.2-EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:
AArch64:

- EL1 reads of CCSIDR EL1, CCSIDR2 EL1, CLIDR EL1, and CSSELR EL1.
- EL1 writes to CSSELR EL1.

AArch32:

- EL1 reads of CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- EL1 writes to CSSELR.

<table>
<thead>
<tr>
<th>TID4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified EL1 and EL0 accesses to ID group 4 registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is \{1, 1\}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [48]

Reserved, RES0.

FIEN, bit [47]

When ARMv8.4-RAS is implemented:

Fault Injection Enable. Unless this bit is set to 1, accesses to the ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18.

<table>
<thead>
<tr>
<th>FIEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to the specified registers from EL1 are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If EL2 is disabled in the current Security state, the Effective value of HCR_EL2.FIEN is 0b1.

If ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FWB, bit [46]

When ARMv8.4-S2FWB is implemented:

Defines the combined cacheability attributes in a 2 stage translation regime.
FWB

<table>
<thead>
<tr>
<th>FWB</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0 | When this bit is 0, then:  
|     | • The combination of stage 1 and stage 2 translations on memory type and cacheability attributes are as described in the Armv8.0 architecture. For more information see D4.5.4 Combining the stage 1 and stage 2 attributes  
|     | • The encoding of the stage 2 memory type and cacheability attributes in bits[5:2] of the stage 2 page or block descriptors are as described in the Armv8.0 architecture. |
| 0b1 | When this bit is 1, then:  
|     | • Bit[5] of stage 2 page or block descriptor is RES0.  
|     | • When bit[4] of stage 2 page or block descriptor is 1 and when:  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b11, the resultant memory type and inner or outer cacheability attribute is the same as the stage 1 memory type and inner or outer cacheability attribute.  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b10, the resultant memory type and attribute is Normal Write-Back.  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b0x, the resultant memory type will be Normal Non-cacheable except where the stage 1 memory type was Device->attr< the resultant memory type will be Device->attr<  
|     | • When bit[4] of stage 2 page or block descriptor is 0 the memory type is Device, and when:  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b00, the stage 2 memory type is Device-nGnRnE.  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b01, the stage 2 memory type is Device-nGnRE.  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b10, the stage 2 memory type is Device-nGRE.  
|     | ◦ Bits[3:2] of stage 2 page or block descriptor are 0b11, the stage 2 memory type is Device-GRE.  
|     | • If the stage 1 translation specifies a cacheable memory type, then the stage 1 cache allocation hint is applied to the final cache allocation hint where the final memory type is cacheable.  
|     | • If the stage 1 translation does not specify a cacheable memory type, then if the final memory type is cacheable, it is treated as read allocate, write allocate.  
|     | The stage 1 and stage 2 memory types are combined in the manner described in D4.5.4 Combining the stage 1 and stage 2 attributes |

In Secure state, this bit applies to both the Secure stage 2 translation and the Non-secure stage 2 translation.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV2, bit [45]

When ARMv8.4-NV is implemented:

Nested Virtualization. Changes the behaviors of HCR_EL2.{NV, NV1} to provide a mechanism for hardware to transform reads and writes from System registers into reads and writes from memory.
TABLE 18-12

<table>
<thead>
<tr>
<th>NV2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit has no effect on the behavior of HCR_EL2.{NV, NV1}.</td>
</tr>
<tr>
<td>0b1</td>
<td>Redefines behavior of HCR_EL2{NV, NV1} to enable:</td>
</tr>
<tr>
<td></td>
<td>• Transformation of read/writes to registers into read/writes</td>
</tr>
<tr>
<td></td>
<td>to memory.</td>
</tr>
<tr>
<td></td>
<td>• Redirection of EL2 registers to EL1 registers.</td>
</tr>
</tbody>
</table>

When this bit is 0, the behavior of HCR_EL2.\{NV, NV1\} is as defined for ARMv8.3-NV.

When this is bit is 1, then any exception taken from EL1 and taken to EL1 causes SPSR_EL1.M[3:2] to be set to 0b10 and not 0b01.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**AT, bit [44]**

**When ARMv8.3-NV is implemented:**

Address Translation. EL1 execution of the following address translation instructions is trapped to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18:

- AT S1E0R, AT S1E0W, AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP

<table>
<thead>
<tr>
<th>AT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 execution of the specified instructions is trapped to EL2.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**NV1, bit [43]**

**When ARMv8.4-NV is implemented:**

Nested Virtualization.

<table>
<thead>
<tr>
<th>NV1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If HCR_EL2.{NV, NV2} are both 1, accesses executed from EL1 to</td>
</tr>
<tr>
<td></td>
<td>implemented EL12, EL02, or EL2 registers are transformed to loads and</td>
</tr>
<tr>
<td></td>
<td>stores.                   If HCR_EL2.NV2 is 0 or HCR_EL2.{NV, NV2} == {0, 1}, this</td>
</tr>
<tr>
<td></td>
<td>control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If HCR_EL2.NV2 is 1, accesses executed from EL1 to implemented EL2</td>
</tr>
<tr>
<td></td>
<td>registers are transformed to loads and stores.</td>
</tr>
<tr>
<td></td>
<td>If HCR_EL2.NV2 is 0, EL1 accesses to VBAR_EL1, ELR_EL1, and</td>
</tr>
<tr>
<td></td>
<td>SPSR_EL1, are trapped to EL2, when EL2 is enabled in the current</td>
</tr>
<tr>
<td></td>
<td>Security state, reported using EC syndrome value 0x18.</td>
</tr>
</tbody>
</table>

If HCR_EL2.NV2 is 1, the value of HCR_EL2.NV1 defines which EL1 register accesses are transformed to loads and stores. These transformed accesses have priority over the trapping of registers.

The trapping of EL1 registers caused by other control bits has priority over the transformation of these accesses.

If a register is specified that is not implemented by an implementation, then access to that register are **UNDEFINED**.

For the list of registers affected, see Enhanced support for nested virtualization.

If HCR_EL2.\{NV, NV1, NV2\} are \{1, 0, 0\}, any exception taken from EL1, and taken to EL1, causes the SPSR_EL1.M[3:2] to be set to 0b10, and not 0b01.
If HCR_EL2.{NV, NV1, NV2} are \{1, 1, 0\}, then:

- The EL1 translation table Block and Page descriptors:
  - Bit[54] holds the PXN instead of the UXN.
  - Bit[53] is RES0.
  - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
  - Bit[61] is treated as 0 regardless of the actual value.
  - Bit[60] holds the PXNTable instead of the UXNTable.
  - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If HCR_EL2.{NV, NV1, NV2} are \{0, 1, 0\}, then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1 for all purposes other than reading back the value of the HCR_EL2.NV bit.
- Behaving as if HCR_EL2.NV is 0 and HCR_EL2.NV1 is 0 for all purposes other than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the HCR_EL2.NV and HCR_EL2.NV1 bits behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

When ARMv8.3-NV is implemented:

Nested Virtualization. EL1 accesses to certain registers are trapped to EL2, when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>NV1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>01</td>
<td>EL1 accesses to VBAR_EL1, ELR_EL1, SPSR_EL1 are trapped to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18.</td>
</tr>
</tbody>
</table>

If HCR_EL2.NV is 1 and HCR_EL2.NV1 is 0 then the following effects also apply:

- Any exception taken from EL1, and taken to EL1, causes the SPSR_EL1.M[3:2] to be set to 0b10, and not 0b01.

If the bits HCR_EL2.NV and HCR_EL2.NV1 are both set to 1 then following effects also apply:

- The EL1 translation table Block and Page descriptors:
  - Bit[54] holds the PXN instead of the UXN.
  - Bit[53] is RES0.
  - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
  - Bit[61] is treated as 0 regardless of the actual value.
  - Bit[60] holds the PXNTable instead of the UXNTable.
  - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If HCR_EL2.NV is 0 and HCR_EL2.NV1 is 1 then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1 for all purposes other than reading back the value of the HCR_EL2.NV bit.
- Behaving as if HCR_EL2.NV is 0 and HCR_EL2.NV1 is 0 for all purposes other than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the HCR_EL2.NV and HCR_EL2.NV1 bits behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.
This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**NV, bit [42]**

*When ARMv8.4-NV is implemented:*

Nested Virtualization.

When HCR_EL2.NV2 is 1, redefines register accesses so that:

- Instructions accessing the Special purpose registers `SPSR_EL2` and `ELR_EL2` instead access `SPSR_EL1` and `ELR_EL1` respectively.
- Instructions accessing the System registers `ESR_EL2` and `FAR_EL2` instead access `ESR_EL1` and `FAR_EL1`.

When HCR_EL2.NV2 is 0, or if ARMv8.4-NV is not implemented, traps functionality that is permitted at EL2 and would be **UNDEFINED** at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not **UNDEFINED** at EL2.
- EL1 accesses to System registers that are not **UNDEFINED** at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

<table>
<thead>
<tr>
<th>NV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When this bit is set to 0, HCR_EL2.NV2 == 0 for all purposes other than reading this register. This control does not cause any instructions to be trapped. When HCR_EL2.NV2 is 1, no ARMv8.4-NV functionality is implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>When HCR_EL2.NV2 is 0, or if ARMv8.4-NV is not implemented, EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the <code>CurrentEL</code> register return a value of 0x2. When HCR_EL2.NV2 is 1, this control redefines EL1 register accesses so that instructions accessing <code>SPSR_EL2</code>, <code>ELR_EL2</code>, <code>ESR_EL2</code>, and <code>FAR_EL2</code> instead access <code>SPSR_EL1</code>, <code>ELR_EL1</code>, <code>ESR_EL1</code>, and <code>FAR_EL1</code> respectively.</td>
</tr>
</tbody>
</table>

When HCR_EL2.NV2 is 0, or if ARMv8.4-NV is not implemented, then:

- The System or Special-purpose registers for which accesses are trapped and reported using EC syndrome value 0x18 are as follows:
  - Registers accessed using MRS or MSR with a name ending in `_EL2`, except `SP_EL2`.
  - Registers accessed using MRS or MSR with a name ending in `_EL12`.
  - Registers accessed using MRS or MSR with a name ending in `_EL02`.
  - Special-purpose registers `SPSR_irq`, `SPSR_abt`, `SPSR_und` and `SPSR_fiq`, accessed using MRS or MSR.
  - Special-purpose register `SP_EL1` accessed using the dedicated MRS or MSR instruction.
- The instructions for which the execution is trapped and reported using EC syndrome value 0x18 are as follows:
  - EL2 translation regime Address Translation instructions and TLB maintenance instructions.
  - EL1 translation regime Address Translation instructions and TLB maintenance instructions that are only accessible from EL2 and EL3.
- The instructions for which the execution is trapped as follows:
  - SMC in an implementation that does not include EL3 and when HCR_EL2.TSC is 1. HCR_EL2.TSC bit is RES0 in this case. This is reported using EC syndrome value 0x17.
  - The ERET, ERETTAA, and ERETAB instructions, reported using EC syndrome value 0x1A.

**Note**

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETTAA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.
This field resets to an architecturally unknown value.

When ARMv8.3-NV is implemented:

Nested Virtualization. Traps functionality that is permitted at EL2 and would be undefined at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not undefined at EL2.
- EL1 accesses to System registers that are not undefined at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

The possible values are:

<table>
<thead>
<tr>
<th>NV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the currentEL register return a value of 0x2.</td>
</tr>
</tbody>
</table>

The System or Special-purpose registers for which accesses are trapped and reported using EC syndrome value 0x18 are as follows:

- Registers accessed using MRS or MSR with a name ending in _EL2, except SP_EL2.
- Registers accessed using MRS or MSR with a name ending in _EL12.
- Registers accessed using MRS or MSR with a name ending in _EL02.
- Special-purpose registers SPSR_irq, SPSR_abt, SPSR_und and SPSR_fiq, accessed using MRS or MSR.
- Special-purpose register SP_EL1 accessed using the dedicated MRS or MSR instruction.

The instructions for which the execution is trapped and reported using EC syndrome value 0x18 are as follows:

- EL2 translation regime Address Translation instructions and TLB maintenance instructions.
- EL1 translation regime Address Translation instructions and TLB maintenance instructions that are only accessible from EL2 and EL3.

The execution of the ERET, ERETA, and ERETAB instructions are trapped and reported using EC syndrome value 0x1A

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.

The execution of the SMC instructions in an implementation that does not include EL3 and when HCR_EL2.TSC is 1 are trapped and reported using EC syndrome value 0x17. HCR_EL2.TSC bit is not res0 in this case.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally unknown value.

Otherwise:

Reserved, res0.

API, bit [41]

When ARMv8.3-PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- In EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M>==1.
- In EL1, the associated SCTLR_EL1.En<N><M>==1.
Traps are reported using EC syndrome value 0x09. The Pointer Authentication instructions trapped are:

- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB.
- PACGA, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB.
- RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETTA, ERETTB, LDRAA and LDRAB.

<table>
<thead>
<tr>
<th>API</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The instructions related to Pointer Authentication are trapped to EL2, when EL2 is enabled in the current Security state and the instructions are enabled for the EL1&amp;0 translation regime, from:</td>
</tr>
<tr>
<td></td>
<td>• EL0 when HCR_EL2.TGE==0 or HCR_EL2.E2H==0.</td>
</tr>
<tr>
<td></td>
<td>• EL1. If HCR_EL2.NV is 1, the HCR_EL2.NV trap takes precedence over</td>
</tr>
<tr>
<td></td>
<td>the HCR_EL2.API trap for the ERETTA and ERETTB instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If ARMv8.3-PAuth is implemented but EL2 is not implemented or disabled in the current Security state, the system behaves as if this bit is 1.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**APK, bit [40]**

**When ARMv8.3-PAuth is implemented:**

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers from EL1 to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18:

- APIAKeyLo_EL1, APIAKeyHi_EL1, APIBKeyLo_EL1, APIBKeyHi_EL1, APDAKeyLo_EL1, APDAKeyHi_EL1, APDBKeyLo_EL1, APDBKeyHi_EL1, APGAKeyLo_EL1, and APGAKeyHi_EL1.

<table>
<thead>
<tr>
<th>APK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to the registers holding &quot;key&quot; values for pointer authentication from EL1 are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

**Note**

If ARMv8.3-PAuth is implemented but EL2 is not implemented or is disabled in the current Security state, the system behaves as if this bit is 1.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TME, bit [39]**

**When TME is implemented:**

Enables access to the TSTART, TCOMMIT, TTEST and TCANCEL instructions at EL0 and EL1.

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 and EL1 accesses to TSTART, TCOMMIT, TTEST and TCANCEL instructions are UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instruction to be UNDEFINED.</td>
</tr>
</tbody>
</table>
If EL2 is not implemented or is disabled in the current Security state, the Effective value of this bit is 0b1.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**MIOCNCE, bit [38]**

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the EL1&0 translation regimes.

<table>
<thead>
<tr>
<th>MIOCNCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For the EL1&amp;0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there must be no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.</td>
</tr>
<tr>
<td>0b1</td>
<td>For the EL1&amp;0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there might be a loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.</td>
</tr>
</tbody>
</table>

For more information see 'Mismatched memory attributes' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section B2 (The AArch64 Application Level Memory Model).

This field can be implemented as RAZ/WI.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally **UNKNOWN** value.

**TEA, bit [37]**

Route synchronous External abort exceptions to EL2. If the RAS Extension is implemented, the possible values of this bit are:

<table>
<thead>
<tr>
<th>TEA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause exceptions to be routed from EL0 and EL1 to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Route synchronous External abort exceptions from EL0 and EL1 to EL2, when EL2 is enabled in the current Security state, if not routed to EL3.</td>
</tr>
</tbody>
</table>

When the RAS Extension is not implemented, this field is RES0.

This field resets to an architecturally **UNKNOWN** value.

**TERR, bit [36]**

When RAS is implemented:

Trap Error record accesses. Trap accesses to the RAS error registers from EL1 to EL2 as follows:

- If EL1 is using AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - **ERRIDR_EL1**, **ERRSETLR_EL1**, **ERXADDR_EL1**, **ERXCTLR_EL1**, **ERXFR_EL1**, **ERXMISC0_EL1**, **ERXMISC1_EL1**, and **ERXSTATUS_EL1**.
  - When ARMv8.4-RAS is implemented, **ERXMISC2_EL1** and **ERXMISC3_EL1**.

- If EL1 is using AArch32 state, MCR or MRRC accesses are trapped to EL2, reported using EC syndrome value 0x03, MCRR or MRRC accesses are trapped to EL2, reported using EC syndrome value 0x04:
HCR_EL2, Hypervisor Configuration Register

- ERRDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTRLR, ERXCTRLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.
- When ARMv8.4-RAS is implemented, ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

<table>
<thead>
<tr>
<th>TERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to the specified registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**TLOR, bit [35]**

When ARMv8.1-LOR is implemented:

Trap LOR registers. Traps accesses to the LORSA_EL1, LOREA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 to EL2, when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>TLOR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to the LOR registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**E2H, bit [34]**

When ARMv8.1-VHE is implemented:

EL2 Host. Enables a configuration where a Host Operating System is running in EL2, and the Host Operating System's applications are running in EL0.

<table>
<thead>
<tr>
<th>E2H</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The facilities to support a Host Operating System at EL2 are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>The facilities to support a Host Operating System at EL2 are enabled.</td>
</tr>
</tbody>
</table>

For information on the behavior of this bit see Behavior of HCR_EL2.E2H.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
ID, bit [33]

Stage 2 Instruction access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

<table>
<thead>
<tr>
<th>ID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on stage 2 of the EL1&amp;0 translation regime.</td>
</tr>
<tr>
<td>0b1</td>
<td>For all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.</td>
</tr>
</tbody>
</table>

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

CD, bit [32]

Stage 2 Data access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

<table>
<thead>
<tr>
<th>CD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on stage 2 of the EL1&amp;0 translation regime for data accesses and translation table walks.</td>
</tr>
<tr>
<td>0b1</td>
<td>For all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.</td>
</tr>
</tbody>
</table>

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

RW, bit [31]

Execution state control for lower Exception levels:

<table>
<thead>
<tr>
<th>RW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Lower levels are all AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Execution state for EL1 is AArch64. The Execution state for EL0 is determined by the current value of PSTATE.nRW when executing at EL0.</td>
</tr>
</tbody>
</table>

If AArch32 state is not supported by the implementation at EL1, then this bit is RAO/WI.

In an implementation that includes EL3, when EL2 is not enabled in Secure state, the PE behaves as if this bit has the same value as the SCR_EL3.RW bit for all purposes other than a direct read or write access of HCR_EL2.

The RW bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this field behaves as 1 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TRVM, bit [30]

Trap Reads of Virtual Memory controls. Traps EL1 reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, from both Execution states as follows:

- If EL1 is using AArch64 state, the following registers are trapped to EL2 and reported using EC syndrome value 0x18.
HCR_EL2, Hypervisor Configuration Register

- SCTLR_EL1, TTBR0_EL1, TTBR1_EL1, TCR_EL1, ESR_EL1, FAR_EL1, AFSR0_EL1, AFSR1_EL1, MAIR_EL1, AMAIR_EL1, CONTEXTIDR_EL1.

- If EL1 is using AArch32 state, accesses using MRC to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, accesses using MRRC are trapped to EL2 and reported using EC syndrome value 0x04:
  - SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIIFSR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.

<table>
<thead>
<tr>
<th>TRVM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 read accesses to the specified Virtual Memory controls are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

**Note**

EL2 provides a second stage of address translation, that a hypervisor can use to remap the address map defined by a Guest OS. In addition, a hypervisor can trap attempts by a Guest OS to write to the registers that control the memory system. A hypervisor might use this trap as part of its virtualization of memory management.

This field resets to an architecturally **UNKNOWN** value.

**HCD, bit [29]**

HVC instruction disable. Disables EL1 execution of HVC instructions, from both Execution states, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x80.

<table>
<thead>
<tr>
<th>HCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>HVC instruction execution is enabled at EL2 and EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>HVC instructions are <strong>UNDEFINED</strong> at EL2 and EL1. Any resulting exception is taken to the Exception level at which the HVC instruction is executed.</td>
</tr>
</tbody>
</table>

**Note**

HVC instructions are always **UNDEFINED** at EL0.

This bit is only implemented if EL3 is not implemented. Otherwise, it is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

**TDZ, bit [28]**

Trap **DC ZVA** instructions. Traps EL0 and EL1 execution of **DC ZVA** instructions to EL2, when EL2 is enabled in the current Security state, from AArch64 state only, reported using EC syndrome value 0x18.

If ARMv8.5-MemTag is implemented, this trap also applies to **DC GVA** and **DC GZVA**.

<table>
<thead>
<tr>
<th>TDZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>In AArch64 state, any attempt to execute an instruction this trap applies to at EL1, or at EL0 when the instruction is not <strong>UNDEFINED</strong> at EL0, is trapped to EL2 when EL2 is enabled in the current Security state. Reading the <strong>DCZID_EL0</strong> returns a value that indicates that the instructions this trap applies to are not supported.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally **UNKNOWN** value.
TGE, bit [27]

Trap General Exceptions, from EL0.

<table>
<thead>
<tr>
<th>TGE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on execution at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, in all cases:</td>
</tr>
<tr>
<td></td>
<td>• All exceptions that would be routed to EL1 are routed to EL2.</td>
</tr>
<tr>
<td></td>
<td>• The SCTLR_EL1.M field, or the SCTLR.M field if EL1 is using AArch32, is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR_EL1 or SCTLR.</td>
</tr>
<tr>
<td></td>
<td>• All virtual interrupts are disabled.</td>
</tr>
<tr>
<td></td>
<td>• Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled.</td>
</tr>
<tr>
<td></td>
<td>• An exception return to EL1 is treated as an illegal exception return.</td>
</tr>
<tr>
<td></td>
<td>• The MDCR_EL2.{TDRA, TDOSA, TDA, TDE} fields are treated as being 1 for all purposes other than returning the result of a direct read of MDCR_EL2.</td>
</tr>
</tbody>
</table>

In addition, when EL2 is enabled in the current Security state, if:

• HCR_EL2.E2H is 0, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 1.
• HCR_EL2.E2H is 1, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 0.

For further information on the behavior of this bit when E2H is 1, see Behavior of HCR_EL2.E2H.

HCR_EL2.TGE must not be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

TVM, bit [26]

Trap Virtual Memory controls. Traps EL1 writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, from both Execution states as follows:

• If EL1 is using AArch64 state, the following registers are trapped to EL2 and reported using EC syndrome value 0x18:
  ◦ SCTLR_EL1, TTBR0_EL1, TTBR1_EL1, TCR_EL1, ESR_EL1, FAR_EL1, AFSR0_EL1, AFSR1_EL1, MAIR_EL1, AMAIR_EL1, CONTEXTIDR_EL1.

• If EL1 is using AArch32 state, accesses using MCR to the following registers are trapped to EL2 and reported using EC syndrome value 0x83, accesses using MCRR are trapped to EL2 and reported using EC syndrome value 0x04:
  ◦ SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IFSR, DFAR, IFAR, ADFSR, AIFSR, PRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.

<table>
<thead>
<tr>
<th>TVM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 write accesses to the specified EL1 virtual memory control registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TTLB, bit [25]

When ARMv8.4-TLBI is implemented:

Trap TLB maintenance instructions. Traps EL1 execution of TLB maintenance instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states as follows:
When EL1 is using AArch64 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x18:

- TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1.
- TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, TLBI VAALE1IS.
- TLBI VMALLE1OS, TLBI VAE1OS, TLBI ASIDE1OS, TLBI VAAE1OS, TLBI VALE1OS, TLBI VAALE1OS.
- TLBI RVAE1, TLBI RVAAE1, TLBI RVALE1, TLBI RVAALE1.
- TLBI RVAE1IS, TLBI RVAAE1IS, TLBI RVALE1IS, TLBI RVAALE1IS.
- TLBI RVAE1OS, TLBI RVAAE1OS, TLBI RVALE1OS, TLBI RVAALE1OS.

When EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x03:

- TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVALIS, TLBIMVAALIS.
- TLBIALL, TLBIMVA, TLBIASID, TLBIMVAL, TLBIMVAAL.
- ITLBIALL, ITLBIASID, ITLBPVAAL.
- DTLBIALL, DTLBIASID, DTLBI VAL.

### TTLB Meaning

<table>
<thead>
<tr>
<th>TTLB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 execution of the specified TLB maintenance instructions are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When \texttt{HCR\_EL2.TGE} is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

**Note**

The TLB maintenance instructions are **undefined** at EL0.

This field resets to an architecturally **unknown** value.

### Otherwise:

Trap TLB maintenance instructions. Traps EL1 execution of TLB maintenance instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states. This applies to the following instructions:

- When EL1 is using AArch64, TLBI VMALLE1IS, TLBI VAE1IS, TLBI ASIDE1IS, TLBI VAAE1IS, TLBI VALE1IS, TLBI VAALE1IS, TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1.
- When EL1 is using AArch32, TLBIALLIS, TLBIMVAIS, TLBIASIDIS, TLBIMVALIS, TLBIMVAALIS, ITLBIALL, ITLBIASID, DTLBIALL, DTLBIASID, TLBIMVAAL.

<table>
<thead>
<tr>
<th>TTLB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 execution of the specified TLB maintenance instructions are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When \texttt{HCR\_EL2.TGE} is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally **unknown** value.

### TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state as follows:

- If EL0 is using AArch64 state and the value of \texttt{SCTLR\_EL1. UCI} is not 0, the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
  - IC IVAU, DC CVAU. If the value of \texttt{SCTLR\_EL1. UCI} is 0 these instructions are **undefined** at EL0 and any resulting exception is higher priority than this trap to EL2.
- If EL1 is using AArch64 state, the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
  - IC IVAU, IC IALLU, IC IALLUIS, DC CVAU.
- If EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
  - TCIMVAU, TC IALLU, TC IALLUIS, DCCMVAU.
An exception generated because an instruction is **UNDEFINED** at EL0 is higher priority than this trap to EL2. In addition:

- **IC IALLUIS** and **IC IALLU** are always **UNDEFINED** at EL0 using AArch64.
- **ICIMVAU, IC IALLU, IC IALLUIS**, and **DCCMVAU** are always **UNDEFINED** at EL0 using AArch32.

<table>
<thead>
<tr>
<th>TPC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

If the Point of Unification is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is **IMPLEMENTATION DEFINED** whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of **HCR_EL2.\{E2H, TGE\}** is \{1, 1\}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally **UNKNOWN** value.

**TPCP, bit [23]**

*When ARMv8.2-DCPoP is implemented:*

Trap data or unified cache maintenance instructions that operate to the Point of Coherency or Persistence. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state as follows:

- If EL0 is using AArch64 state and the value of **SCTLR_EL1.\text{UCI}** is not 0, the following instructions are trapped to EL2 and reported using EC syndrome value 0x18:
  - **DC CIVAC, DC CVAC, DC CVAP.** If the value of **SCTLR_EL1.\text{UCI}** is 0 these instructions are **UNDEFINED** at EL0 and any resulting exception is higher priority than this trap to EL2.
- If EL1 is using AArch64 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x18:
  - **DC IVAC, DC CIVAC, DC CVAC, DC CVAP.**
- If EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x03:
  - **DCIMVAC, DCCIMVAC, DCCMVAC.**

If ARMv8.2-DCCVADP is implemented, this trap also applies to **DC CVADP**.

If ARMv8.5-MemTag is implemented, this trap also applies to **DC CIGVAC, DC CIGDVAC, DC IGVAC, DC IGDVAC, DC CGVAC, DC CGDVAC, DC CGVAP**, and **DC CGDVAP**.

If ARMv8.2-DCCVADP and ARMv8.5-MemTag are implemented, this trap also applies to **DC CGVADP** and **DC CGDVADP**.

An exception generated because an instruction is **UNDEFINED** at EL0 is higher priority than this trap to EL2. In addition:

- AArch64 instructions which invalidate by VA to the Point of Coherency are always **UNDEFINED** at EL0 using AArch64.
- **DCIMVAC, DCCIMVAC**, and **DCCMVAC** are always **UNDEFINED** at EL0 using AArch32.
- In Armv8.0 and Armv8.1, this field is named TPC. From Armv8.2 it is named TPCP.

<table>
<thead>
<tr>
<th>TPC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>
If the Point of Coherency is before any level of data cache, it is implementation defined whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If $HCR\_EL2\{E2H, TGE\}$ is set to $\{1, 1\}$, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally unknown value.

**Otherwise:**

Trap data or unified cache maintenance instructions that operate to the Point of Coherency. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state as follows:

- If EL0 is using AArch64 state and the value of $SCTL\_EL1\_UCI$ is not 0, accesses to the following registers are trapped and reported using EC syndrome value $0x18$:
  - $DC\_CIVAC$, $DC\_CVAC$. However, if the value of $SCTL\_EL1\_UCI$ is 0 these instructions are undefined at EL0 and any resulting exception is higher priority than this trap to EL2.
- If EL1 is using AArch64 state, accesses to $DC\_IVAC$, $DC\_CIVAC$, $DC\_CVAC$ are trapped and reported using EC syndrome value $0x18$.
- When EL1 is using AArch32, accesses to $DCIMVAC$, $DCCIMVAC$, and $DCCMVAC$ are trapped and reported using EC syndrome value $0x03$.

**Note**

- An exception generated because an instruction is undefined at EL0 is higher priority than this trap to EL2. In addition:
  - AArch64 instructions which invalidate by VA to the Point of Coherency are always undefined at EL0 using AArch64.
  - $DCIMVAC$, $DCCIMVAC$, and $DCCMVAC$ are always undefined at EL0 using AArch32.
- In Armv8.0 and Armv8.1, this field is named TPC. From Armv8.2 it is named TPCP.

<table>
<thead>
<tr>
<th>TPC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

If the Point of Coherency is before any level of data cache, it is implementation defined whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of $HCR\_EL2\{E2H, TGE\}$ is $\{1, 1\}$, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally unknown value.

**TSW, bit [22]**

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps execution of those cache maintenance instructions at EL1 using AArch64, and at EL1 using AArch32, to EL2 when EL2 is enabled in the current Security state as follows:

- If EL1 is using AArch64 state, accesses to $DC\_ISW$, $DC\_CSW$, $DC\_CISW$ are trapped to EL2, reported using EC syndrome value $0x18$.
- If EL1 is using AArch32 state, accesses to $DCISW$, $DCCSW$, $DCCISW$ are trapped to EL2, reported using EC syndrome value $0x03$.

If ARMv8.5-MemTag is implemented, this trap also applies to $DC\_IGSW$, $DC\_IGDSW$, $DC\_CGSW$, $DC\_CGDW$, $DC\_CIGSW$, and $DC\_CIGDSW$.

**Note**

An exception generated because an instruction is undefined at EL0 is higher priority than this trap to EL2, and these instructions are always undefined at EL0.
When `HCR_EL2.TGE` is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally `UNKNOWN` value.

**TACR, bit [21]**

Trap Auxiliary Control Registers. Traps EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, from both Execution states as follows:

- If EL1 is using AArch64 state, accesses to `ACTLR_EL1` to EL2, are trapped to EL2 and reported using EC syndrome value 0x18.
- If EL1 is using AArch32 state, accesses to `ACTLR` and, if implemented, `ACTLR2` are trapped to EL2 and reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>TACR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to the specified registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When `HCR_EL2.TGE` is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

---

**Note**

`ACTLR_EL1`, `ACTLR`, and `ACTLR2` are not accessible at EL0.

The Auxiliary Control Registers are **IMPLEMENTATION DEFINED** registers that might implement global control bits for the PE.

This field resets to an architecturally `UNKNOWN` value.

---

**TIDCP, bit [20]**

Trap **IMPLEMENTATION DEFINED** functionality. Traps EL1 accesses to the encodings reserved for **IMPLEMENTATION DEFINED** functionality to EL2, when EL2 is enabled in the current Security state as follows:

- In AArch64 state, access to any of the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome 0x18:
  - **IMPLEMENTATION DEFINED** System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}.
  - **IMPLEMENTATION DEFINED** System registers, which are accessed using MRS and MSR with the S3_<op1>_<_Cn>_<_Cm>_<_op2> register name.
- In AArch32 state, MCR and MRC access to instructions with the following encodings are trapped and reported using EC syndrome 0x03:
  - All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}.
  - All coproc==p15, CRn==c10, opc1 == {0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}.
  - All coproc==p15, CRn==c11, opc1 == {0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

When the value of HCR_EL2.TIDCP is 1, it is **IMPLEMENTATION DEFINED** whether any of this functionality accessed from EL0 is trapped to EL2. If it is not, then it is **UNDEFINED**, and any attempt to access it from EL0 generates an exception that is taken to EL1.

<table>
<thead>
<tr>
<th>TIDCP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to or execution of the specified encodings reserved for <strong>IMPLEMENTATION DEFINED</strong> functionality are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

An implementation can also include **IMPLEMENTATION DEFINED** registers that provide additional controls, to give finer-grained control of the trapping of **IMPLEMENTATION DEFINED** features.

---

**Note**
Arm expects the trapping of EL0 accesses to these functions to EL2 to be unusual, and used only when the hypervisor is virtualizing EL0 operation. Arm strongly recommends that unless the hypervisor must virtualize EL0 operation, an EL0 access to any of these functions is **UNDEFINED**, as it would be if the implementation did not include EL2. The PE then takes any resulting exception to EL1.

The trapping of accesses to these registers from EL1 is higher priority than an exception resulting from the register access being **UNDEFINED**.

This field resets to an architecturally **UNKNOWN** value.

### TSC, bit [19]

Trap SMC instructions. Traps EL1 execution of SMC instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states.

If execution is in AArch64 state the trap is reported using EC syndrome value 0x17 and if execution is in AArch32 state, the trap is reported using EC syndrome value 0x13.

**Note**

HCR_EL2.TSC traps execution of the SMC instruction. It is not a routing control for the SMC exception. Trap exceptions and SMC exceptions have different preferred return addresses.

<table>
<thead>
<tr>
<th>TSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>
| 0b1 | If EL3 is implemented, then any attempt to execute an SMC instruction at EL1 using AArch64 or EL1 using AArch32 is trapped to EL2, when EL2 is enabled in the current Security state, regardless of the value of SCR_EL3.SMD. If EL3 is not implemented, ARMv8.3-NV is implemented, and HCR_EL2.NV is 1, then any attempt to execute an SMC instruction at EL1 using AArch64 is trapped to EL2, when EL2 is enabled in the current Security state. If EL3 is not implemented, and either ARMv8.3-NV is not implemented or HCR_EL2.NV is 0, then it is IMPLEMENTATION DEFINED whether:  
• Any attempt to execute an SMC instruction at EL1 is trapped to EL2, when EL2 is enabled in the current Security state.  
• Any attempt to execute an SMC instruction is **UNDEFINED**. |

In AArch32 state, the Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions. SMC instructions are **UNDEFINED** at EL0.

If EL3 is not implemented and HCR_EL2.NV is 0, it is IMPLEMENTATION DEFINED whether this bit is:  

• RES0.  
• Implemented with the functionality as described in HCR_EL2.TSC.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field. This field resets to an architecturally **UNKNOWN** value.

### TID3, bit [18]

Trap ID group 3. Traps EL1 reads of group 3 ID registers to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, reads of the following registers are trapped to EL2, reported using EC syndrome value 0x18:
In AArch64 state, \texttt{ID.MMFR4.EL1} and \texttt{ID.MMFR5.EL1} are trapped to EL2, unless implemented as RAZ, when it is \texttt{IMPLEMENTATION DEFINED} whether accesses to \texttt{ID.MMFR4.EL1} or \texttt{ID.MMFR5.EL1} are trapped to EL2, reported using EC syndrome value 0x18.

In AArch64 state, \texttt{ID.MMFR2.EL1} and \texttt{ID.ISAR6.EL1} are trapped to EL2, unless implemented as RAZ, when it is \texttt{IMPLEMENTATION DEFINED} whether accesses to \texttt{ID.MMFR2.EL1} or \texttt{ID.ISAR6.EL1} are trapped to EL2, reported using EC syndrome value 0x18.

In AArch64 state, \texttt{ID.DFR1.EL1} is trapped to EL2, unless implemented as RAZ, when it is \texttt{IMPLEMENTATION DEFINED} whether accesses to \texttt{ID.DFR1.EL1} are trapped to EL2, reported using EC syndrome value 0x18.

In AArch64 state, \texttt{ID.MMFR4.EL1} is trapped to EL2, unless implemented as RAZ then it is \texttt{IMPLEMENTATION DEFINED} whether accesses to \texttt{ID.MMFR4.EL1} are trapped to EL2, reported using EC syndrome value 0x18.

In AArch64 state, it is \texttt{IMPLEMENTATION DEFINED} whether this field traps MRS accesses to encodings in the following range that are not already mentioned in this field description, reported using EC syndrome value 0x18:
\begin{itemize}
  \item Op0 == 3, op1 == 0, CRn == c0, CRm == \{c2-c7\}, op2 == \{0-7\}.
\end{itemize}

In AArch32 state, MRC access to the following registers are trapped to EL2, reported using EC syndrome value 0x03:
\begin{itemize}
  \item \texttt{ID.MMFR4}, \texttt{ID.MMFR5}, \texttt{ID.MVFR0}, \texttt{MVFR1}, and \texttt{MVFR2}, reported using EC syndrome value 0x08.
\end{itemize}

In AArch32 state, \texttt{ID.ISAR6} is trapped to EL2, unless implemented as RAZ, when it is \texttt{IMPLEMENTATION DEFINED} whether accesses to \texttt{ID.ISAR6} are trapped to EL2, reported using EC syndrome value 0x03.

In AArch32 state, \texttt{ID.DFR1} is trapped to EL2, unless implemented as RAZ, when it is \texttt{IMPLEMENTATION DEFINED} whether accesses to \texttt{ID.DFR1} are trapped to EL2, reported using EC syndrome value 0x03.

In AArch32 state, it is \texttt{IMPLEMENTATION DEFINED} whether this bit traps MRS accesses to encodings that are not already mentioned, with coproc==p15, opc1 == 0, CRn == c0, CRm == \{c2-c7\}, opc2 == \{0-7\}.

\begin{table}[h]
\centering
\begin{tabular}{|c|l|}
\hline
\textbf{TID3} & \textbf{Meaning} \\
\hline
0b0 & This control does not cause any instructions to be trapped. \\
0b1 & The specified EL1 read accesses to ID group 3 registers are trapped to EL2, when EL2 is enabled in the current Security state. \\
\hline
\end{tabular}
\end{table}

When \texttt{HCR_EL2.TGE} is 1, the PE ignores the value of this field for all purposes other than a direct read of this field. This field resets to an architecturally \texttt{UNKNOWN} value.

\textbf{TID2, bit [17]}

Trap ID group 2. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state, as follows:

\begin{itemize}
  \item If EL1 is using AArch64, reads of \texttt{CTR_EL0}, \texttt{CCSIDR_EL1}, \texttt{CCSIDR2_EL1}, \texttt{CLIDR_EL1}, and \texttt{CSSELR_EL1} are trapped to EL2, reported using EC syndrome value 0x18.
\end{itemize}
If EL0 is using AArch64 and the value of SCTLR_EL1 UCT is not 0, reads of CTR_EL0 are trapped to EL2, reported using EC syndrome value 0x18. If the value of SCTLR_EL1 UCT is 0 then EL0 reads of CTR_EL0 are UNDEFINED and any resulting exception takes precedence over this trap.

If EL1 is using AArch64, writes to CSSELR_EL1 are trapped to EL2, reported using EC syndrome value 0x18.

If EL1 is using AArch32, reads of CTR, CCSIDR, CCSIDR2, CLIDR, and CSSELR are trapped to EL2, reported using EC syndrome value 0x03.

If EL1 is using AArch32, writes to CSSELR are trapped to EL2, reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>TID2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified EL1 and EL0 accesses to ID group 2 registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

TID1, bit [16]

Trap ID group 1. Traps EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state as follows:

- **In AArch64 state**, accesses of REVIDR_EL1, AIDR_EL1, reported using EC syndrome value 0x18.
- **In AArch32 state**, accesses of TCMTR, TLBTR, REVIDR, AIDR, reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>TID1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified EL1 read accesses to ID group 1 registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

TID0, bit [15]

Trap ID group 0. Traps the following register accesses to EL2:

- EL1 reads of the JIDR, reported using EC syndrome value 0x05.
- If the JIDR is RAZ from EL0, EL0 reads of the JIDR, reported using EC syndrome value 0x05.
- EL1 accesses using VMRS of the FPSID, reported using EC syndrome value 0x08.

**Note**

- It is IMPLEMENTATION DEFINED whether the JIDR is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0 then any resulting exception takes precedence over this trap.
- The FPSID is not accessible at EL0 using AArch32.
- Writes to the FPSID are ignored, and not trapped by this control.

<table>
<thead>
<tr>
<th>TID0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified EL1 read accesses to ID group 0 registers are trapped to EL2, when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

In an AArch64 only implementation, this bit is RES0.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.
**TWE, bit [14]**

Traps EL0 and EL1 execution of WFE instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states, reported using EC syndrome value 0x01.

<table>
<thead>
<tr>
<th>TWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFE instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE or SCTLR_EL1.nTWE.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally unknown value.

**TWI, bit [13]**

Traps EL0 and EL1 execution of WFI instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states, reported using EC syndrome value 0x01.

<table>
<thead>
<tr>
<th>TWI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFI instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI or SCTLR_EL1.nTWI.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally unknown value.
Default Cacheability.

<table>
<thead>
<tr>
<th>DC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the EL1&amp;0 translation regime.</td>
</tr>
<tr>
<td>0b1</td>
<td>In both Security states:</td>
</tr>
<tr>
<td></td>
<td>• When EL1 is using AArch64, the PE behaves as if the value of the</td>
</tr>
<tr>
<td></td>
<td>SCTL_EL1.M field is 0 for all purposes other than returning the</td>
</tr>
<tr>
<td></td>
<td>value of a direct read of SCTL_EL1.</td>
</tr>
<tr>
<td></td>
<td>• When EL1 is using AArch32, the PE behaves as if the value of the</td>
</tr>
<tr>
<td></td>
<td>SCTL_EL1.M field is 0 for all purposes other than returning the</td>
</tr>
<tr>
<td></td>
<td>value of a direct read of SCTL.</td>
</tr>
<tr>
<td></td>
<td>• The PE behaves as if the value of the HCR_EL2.VM field is 1 for all</td>
</tr>
<tr>
<td></td>
<td>purposes other than returning the value of a direct read of HCR_EL2.</td>
</tr>
<tr>
<td></td>
<td>• The memory type produced by stage 1 of the EL1&amp;0 translation regime</td>
</tr>
<tr>
<td></td>
<td>is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-</td>
</tr>
<tr>
<td></td>
<td>Allocate, Outer Write-Back Read-Allocate Write-Allocate.</td>
</tr>
</tbody>
</table>

This field has no effect on the EL2, EL2&0, and EL3 translation regimes.

This field is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this field.

This field resets to an architecturally UNKNOWN value.

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from EL1 or EL0:

<table>
<thead>
<tr>
<th>BSU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Full system.</td>
</tr>
</tbody>
</table>

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0b00 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from EL1:

AArch32: BPIALL, TLBIALL, TLBIMVA, TLBISAD, DTLBIALL, DTLBIMVA, DTLBISAD, TLBIALL, TLTBIMVA, ITLBISAD, TLBIMVAA, ICIALL, TLBIVAL, TLBIVALA.

AArch64: TLBI VMALLE1, TLBI VAE1, TLBI ASIDE1, TLBI VAAE1, TLBI VALE1, TLBI VAALE1, IC IALLU, TLBI RVAE1, TLBI RVAE1, TLBI RVALE1, TLBI VRAALE1.

<table>
<thead>
<tr>
<th>FB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This field has no effect on the operation of the specified instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>When one of the specified instruction is executed at EL1, the instruction is broadcast within the Inner Shareable shareability domain.</td>
</tr>
</tbody>
</table>

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.
This field resets to an architecturally **UNKNOWN** value.

**VSE, bit [8]**

Virtual SError interrupt.

<table>
<thead>
<tr>
<th>VSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This mechanism is not making a virtual SError interrupt pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>A virtual SError interrupt is pending because of this mechanism.</td>
</tr>
</tbody>
</table>

The virtual SError interrupt is only enabled when the value of HCR_EL2.{TGE, AMO} is \(\{0, 1\}\).

This field resets to an architecturally **UNKNOWN** value.

**VI, bit [7]**

Virtual IRQ Interrupt.

<table>
<thead>
<tr>
<th>VI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This mechanism is not making a virtual IRQ pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>A virtual IRQ is pending because of this mechanism.</td>
</tr>
</tbody>
</table>

The virtual IRQ is enabled only when the value of HCR_EL2.{TGE, IMO} is \(\{0, 1\}\).

This field resets to an architecturally **UNKNOWN** value.

**VF, bit [6]**

Virtual FIQ Interrupt.

<table>
<thead>
<tr>
<th>VF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This mechanism is not making a virtual FIQ pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>A virtual FIQ is pending because of this mechanism.</td>
</tr>
</tbody>
</table>

The virtual FIQ is enabled only when the value of HCR_EL2.{TGE, FMO} is \(\{0, 1\}\).

This field resets to an architecturally **UNKNOWN** value.

**AMO, bit [5]**

Physical SError interrupt routing.

<table>
<thead>
<tr>
<th>AMO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When executing at Exception levels below EL2, and EL2 is enabled in the current Security state:</td>
</tr>
<tr>
<td></td>
<td>• Physical SError interrupts are not taken to EL2.</td>
</tr>
<tr>
<td></td>
<td>• When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical SError interrupts are not taken unless they are routed to EL3 by the SCR_EL3.EA bit.</td>
</tr>
<tr>
<td></td>
<td>• Virtual SError interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>When executing at any Exception level, and EL2 is enabled in the current Security state:</td>
</tr>
<tr>
<td></td>
<td>• Physical SError interrupts are taken to EL2, unless they are routed to EL3.</td>
</tr>
<tr>
<td></td>
<td>• When the value of HCR_EL2.TGE is 0, then virtual SError interrupts are enabled.</td>
</tr>
</tbody>
</table>

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the AMO bit physical asynchronous External aborts and SError interrupts target EL2 unless they are routed to EL3.
- When ARMv8.1-VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.
HCR_EL2, Hypervisor Configuration Register

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

This field resets to an architecturally UNKNOWN value.

**IMO, bit [4]**

Physical IRQ Routing.

<table>
<thead>
<tr>
<th>IMO</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0 | When executing at Exception levels below EL2, and EL2 is enabled in the current Security state:  
  • Physical IRQ interrupts are not taken to EL2.  
  • When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical IRQ interrupts are not taken unless they are routed to EL3 by the SCR_EL3.IIRQ bit.  
  • Virtual IRQ interrupts are disabled. |
| 0b1 | When executing at any Exception level, and EL2 is enabled in the current Security state:  
  • Physical IRQ interrupts are taken to EL2, unless they are routed to EL3.  
  • When the value of HCR_EL2.TGE is 0, then Virtual IRQ interrupts are enabled. |

If EL2 is enabled in the current Security state, and the value of HCR_EL2.TGE is 1:

• Regardless of the value of the IMO bit, physical IRQ Interrupts target EL2 unless they are routed to EL3.
• When ARMv8.1-VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
• When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

**FMO, bit [3]**

Physical FIQ Routing.

<table>
<thead>
<tr>
<th>FMO</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0 | When executing at Exception levels below EL2, and EL2 is enabled in the current Security state:  
  • Physical FIQ interrupts are not taken to EL2.  
  • When the value of HCR_EL2.TGE is 0, if the PE is executing at EL2 using AArch64, physical FIQ interrupts are not taken unless they are routed to EL3 by the SCR_EL3.FIQ bit.  
  • Virtual FIQ interrupts are disabled. |
| 0b1 | When executing at any Exception level, and EL2 is enabled in the current Security state:  
  • Physical FIQ interrupts are taken to EL2, unless they are routed to EL3.  
  • When HCR_EL2.TGE is 0, then Virtual FIQ interrupts are enabled. |

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

• Regardless of the value of the FMO bit, physical FIQ Interrupts target EL2 unless they are routed to EL3.
• When ARMv8.1-VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
• When ARMv8.1-VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.
PTW, bit [2]

Protected Table Walk. In the EL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the two stages of translation means the access might be made to a type of Device memory. If this occurs, then the value of this bit determines the behavior:

<table>
<thead>
<tr>
<th>PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively.</td>
</tr>
<tr>
<td>0b1</td>
<td>The memory access generates a stage 2 Permission fault.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field. This field resets to an architecturally UNKNOWN value.

SWIO, bit [1]

Set/Way Invalidation Override. Causes EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way:

<table>
<thead>
<tr>
<th>SWIO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the operation of data cache invalidate by set/way instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way.</td>
</tr>
</tbody>
</table>

When the value of this bit is 1:

AArch32: DCISW performs the same invalidation as a DCCISW instruction.

AArch64: DCISW performs the same invalidation as a DCISW instruction.

This bit can be implemented as RES1.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

This field resets to an architecturally UNKNOWN value.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the EL1&0 translation regime, when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>VM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1&amp;0 stage 2 address translation disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1&amp;0 stage 2 address translation enabled.</td>
</tr>
</tbody>
</table>

When the value of this bit is 1, data cache invalidate instructions executed at EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR_EL2.SWIO bit.

This bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2. {E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

Accessing the HCR_EL2

Accesses to this register use the following encodings:
MRS <Xt>, HCR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x078];
  elseif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  return HCR_EL2;
elsif PSTATE.EL == EL3 then
  return HCR_EL2;

MSR HCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x078] = X[t];
  elseif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  HCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  HCR_EL2 = X[t];
The HDFGRTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MRS and MRC reads of debug, trace, PMU, and Statistical Profiling System registers.

**Configuration**

This register is present only when ARMv8.6-FGT is implemented. Otherwise, direct accesses to HDFGRTR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HDFGRTR_EL2 is a 64-bit register.

**Field descriptions**

The HDFGRTR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>PMUSERENR_EL0</td>
</tr>
<tr>
<td>56-55</td>
<td>TRBTRG_EL1, TRBSR</td>
</tr>
<tr>
<td>31-23</td>
<td>PMISRR_EL1, PMISDR_EL1, PMISCR_EL1, PMSFCR_EL1, PMSEVFR_EL1, PMSCR_EL1, PMBSR_EL1, PMBPTR_EL1, PMBLIM</td>
</tr>
</tbody>
</table>

**Bits [63:58]**

Reserved, RES0.

**PMUSERENR_EL0, bit [57]**

When PMUv3 is implemented:

Trap MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMUSERENR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMUSERENR_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMUSERENR at EL0 using AArch32 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads of PMUSERENR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**TRBTRG_EL1, bit [56]**

*When TRBE is implemented:*

Trap MRS reads of TRBTRG_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBTRG_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBTRG_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBTRG_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**TRBSR_EL1, bit [55]**

*When TRBE is implemented:*

Trap MRS reads of TRBSR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBSR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBSR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**TRBPTR_EL1, bit [54]**

*When TRBE is implemented:*

Trap MRS reads of TRBPTR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBPTR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBPTR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TRBMAR_EL1, bit [53]**

When TRBE is implemented:

Trap MRS reads of TRBMAR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBMAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBMAR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TRBLIMITR_EL1, bit [52]**

When TRBE is implemented:

Trap MRS reads of TRBLIMITR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBLIMITR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBLIMITR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TRBIDR_EL1, bit [51]**

When TRBE is implemented:

Trap MRS reads of TRBIDR_EL1 at EL1 using AArch64 to EL2.
HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register

<table>
<thead>
<tr>
<th>TRBIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBIDR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBBASER_EL1, bit [50]

When TRBE is implemented:

Trap MRS reads of TRBBASER_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBBASER_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRBBASER_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRBBASER_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [49]

Reserved, RES0.

TRCVICTLR, bit [48]

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of TRCVICTLR at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCVICTLR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCVICTLR are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRCVICTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
**TRCSTATR, bit [47]**

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of **TRCSTATR** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCSTATR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>TRCSTATR</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>TRCSTATR</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
<tr>
<td>0b1</td>
<td>MRS reads of <strong>TRCSTATR</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>TRCSTATR</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCSSCSRn, bit [46]**

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of **TRCSSCSR<n>** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCSSCSRn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>TRCSSCSR&lt;n&gt;</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>TRCSSCSR&lt;n&gt;</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
<tr>
<td>0b1</td>
<td>MRS reads of <strong>TRCSSCSR&lt;n&gt;</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>TRCSSCSR&lt;n&gt;</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If Single-shot Comparator n is not implemented, a read of **TRCSSCSR<n>** is UNDEFINED.

This bit is RES0 if **TRCSSCSR<n>** are not implemented.

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCSEQSTR, bit [45]**

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of **TRCSEQSTR** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCSEQSTR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>TRCSEQSTR</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>TRCSEQSTR</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
<tr>
<td>0b1</td>
<td>MRS reads of <strong>TRCSEQSTR</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>TRCSEQSTR</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

This bit is RES0 if **TRCSEQSTR** is not implemented.

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:
Reserved, RES0.

**TRCPRGCTLR, bit [44]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MRS reads of TRCPRGCTLR at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCPRGCTLR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCPRGCTLR are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRCPRGCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**TRCOSLSR, bit [43]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MRS reads of TRCOSLSR at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCOSLSR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCOSLSR are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRCOSLSR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**Bit [42]**

Reserved, RES0.

**TRCIMSPEC<n>, bit [41]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MRS reads of TRCIMSPEC<n> at EL1 using AArch64 to EL2.
### TRCIMSPECn

<table>
<thead>
<tr>
<th>TRCIMSPECn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCIMSPEC&lt;n&gt; are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRCIMSPEC&lt;n&gt; at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

TRCIMSPEC<1-7> are optional. If TRCIMSPEC<n> is not implemented, a read of TRCIMSPEC<n> is **UNDEFINED**.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

### TRCID, bit [40]

#### When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TRCDEVARCH.
- TRCDEVID.
- TRCIDR<n>.

### TRCID

<table>
<thead>
<tr>
<th>TRCID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

### Bits [39:38]

Reserved, RES0.

### TRCCNTVRn, bit [37]

#### When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of TRCCNTVR<n> at EL1 using AArch64 to EL2.

### TRCCNTVRn

<table>
<thead>
<tr>
<th>TRCCNTVRn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCCNTVR&lt;n&gt; are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRCCNTVR&lt;n&gt; at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If Counter n is not implemented, a read of TRCCNTVR<n> is **UNDEFINED**.
This bit is RES0 if TRCCNTVR<\text{n}> are not implemented.

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCCLAIM, bit [36]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:* 

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TRCCLAIMCLR.
- TRCCLAIMSET.

<table>
<thead>
<tr>
<th>TRCCLAIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or ( \text{SCR_EL3.FGTEn} == 1 ), MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCAUXCTL, bit [35]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:* 

Trap MRS reads of TRCAUXCTL at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCAUXCTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCAUXCTL are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or ( \text{SCR_EL3.FGTEn} == 1 ), MRS reads of TRCAUXCTL at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCAUTHSTATUS, bit [34]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:* 

Trap MRS reads of TRCAUTHSTATUS at EL1 using AArch64 to EL2.
TRCAUTHSTATUS

| Meaning | TRCAUTHSTATUS
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TRCAUTHSTATUS are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TRCAUTHSTATUS at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRC, bit [33]

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TRCACATR<n>.
- TRCACVR<n>.
- TRCBBCTLR.
- TRCCCCTLRL.
- TRCCIDCCTLR<n>.
- TRCCIDCVR<n>.
- TRCCNCTLR<n>.
- TRCCNTLDR<n>.
- TRCCONFIGR.
- TRCEVENTCTRL0R.
- TRCEVENTCTRL1R.
- TRCEXTPINSELRL<n>, if ETE is implemented
- TRCEXTPINSELRL, if ETMv4 is implemented.
- TRCQCTRL.
- TRCRSCTRLR<n>.
- TCRSR, if ETE is implemented.
- TRCSEQEVR<n>.
- TRCSEQRSTEVRE.
- TRCSSCCR<n>.
- TRCSSFCICR<n>.
- TRCSTALLCTRLR.
- TRCSYNCPVR.
- TRCTRACEIDR.
- TRCTSCTRLR.
- TRCVIIECTLR.
- TRCVIPCSSCTRLR.
- TRCVISSCTRLR.
- TRCVMMIDCCTLR<n>.
- TRCVMMIDCVR<n>.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>TRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

A read of an unimplemented register is UNDEFINED.

TRCEXTPINSELRL<n> and TRCRSR are only implemented if ETE is implemented.

TRCEXTPINSELRL is only implemented if ETE is not implemented and ETMv4 is implemented.
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSLATFR_EL1, bit [32]**

When SPE is implemented:

Trap MRS reads of **PMSLATFR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSLATFR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>PMSLATFR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>PMSLATFR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSIRR_EL1, bit [31]**

When SPE is implemented:

Trap MRS reads of **PMSIRR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSIRR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>PMSIRR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>PMSIRR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSIDR_EL1, bit [30]**

When SPE is implemented:

Trap MRS reads of **PMSIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>PMSIDR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>PMSIDR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMSICR_EL1, bit [29]**

When SPE is implemented:

Trap MRS reads of **PMSICR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSICR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>PMSICR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>PMSICR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMSFCR_EL1, bit [28]**

When SPE is implemented:

Trap MRS reads of **PMSFCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSFCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>PMSFCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>PMSFCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMSEVFR_EL1, bit [27]**

When SPE is implemented:

Trap MRS reads of **PMSEVFR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSEVFR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>PMSEVFR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>PMSEVFR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSCR_EL1, bit [26]**

*When SPE is implemented:*

Trap MRS reads of PMSCR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of PMSCR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of PMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMBSR_EL1, bit [25]**

*When SPE is implemented:*

Trap MRS reads of PMBSR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMBSR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of PMBSR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of PMBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMBPTR_EL1, bit [24]**

*When SPE is implemented:*

Trap MRS reads of PMBPTR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMBPTR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of PMBPTR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of PMBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBLIMITR_EL1, bit [23]

When SPE is implemented:

Trap MRS reads of PMBLIMITR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMBLIMITR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of PMBLIMITR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of PMBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMMIR_EL1, bit [22]

When PMUv3 is implemented:

Trap MRS reads of PMMIR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMMIR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of PMMIR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of PMMIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [21:20]

Reserved, RES0.

PMSELR_EL0, bit [19]

When PMUv3 is implemented:

Trap MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMSELR at EL0 using AArch32 when EL1 is using AArch64 to EL2.
### PMSELR_EL0

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0</th>
<th>0b1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMSELR at EL0 using AArch32 are not affected by this bit.</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2 (E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• MRC reads of PMSELR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.</td>
<td></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### PMOVS, bit [18]

When PMUv3 is implemented:

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of PMOVSCCLR_EL0 and PMOVSET_EL0.
- At EL0 using Arch32 when EL1 is using AArch64: MRC reads of PMOVSR and PMOVSET.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0</th>
<th>0b1</th>
</tr>
</thead>
<tbody>
<tr>
<td>The operations listed above are not affected by this bit.</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2 (E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• MRS reads at EL1 and EL0 using AArch64 of PMOVSCCLR_EL0 and PMOVSET_EL0 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• MRC reads at EL0 using AArch32 of PMOVSR and PMOVSET are trapped to EL2 and reported with EC syndrome value 0x03, unless the read generates a higher priority exception.</td>
<td></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### PMINTEN, bit [17]

When PMUv3 is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- PMINTENCLR_EL1.
- PMINTENSET_EL1.
PMINTEN

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMINTEN, bit [16]

When PMUv3 is implemented:

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of PMCNTENCLR_EL0 and PMCNTENSET_EL0.
- At EL0 using Arch32 when EL1 is using AArch64: MRC reads of PMCNTENCLR and PMCNTENSET.

PMINTEN

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
</tbody>
</table>
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 FGTEn == 1, then, unless the read generates a higher priority exception:
  - MRS reads at EL1 and EL0 using AArch64 of PMCNTENCLR_EL0 and PMCNTENSET_EL0 are trapped to EL2 and reported with EC syndrome value 0x18.
  - MRC reads at EL0 using AArch32 of PMCNTENCLR and PMCNTENSET are trapped to EL2 and reported with EC syndrome value 0x03, unless the read generates a higher priority exception. |

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCCNTR_EL0, bit [15]

When PMUv3 is implemented:

Trap MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MRC and MRRc reads of PMCCNTR at EL0 using AArch32 when EL1 is using AArch64 to EL2.
PMCCNTR_EL0

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0: MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MRC and MRRC reads of PMCCNTR at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1: If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:
  | • MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. |
  | • MRC and MRRC reads of PMCCNTR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03 (for MRC) or 0x04 (for MRRC). |

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCCFILTR_EL0, bit [14]

When PMUv3 is implemented:

Trap MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCCFILTR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0: MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCCFILTR at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1: If EL2 is implemented and enabled in the current Security state, HCR_EL2.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:
  | • MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. |
  | • MRC reads of PMCCFILTR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

PMCCFILTR_EL0 can also be accessed in AArch64 state using PMXEVTYPER_EL0 when PMSELR_EL0 SEL == 31, and PMCCFILTR can also be accessed in AArch32 state using PMXEVTYPER when PMSELR.SEL == 31.

Setting this bit to 1 has no effect on accesses to PMXEVTYPER_EL0 and PMXEVTYPER, regardless of the value of PMSELR_EL0 or PMSELR.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMEVTPYPERn_EL0, bit [13]

When PMUv3 is implemented:

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:
At EL1 and EL0 using AArch64: MRS reads of PMEVTYPER<\n>_EL0 and PMXEVTYPER_EL0.

At EL0 using Arch32 when EL1 is using AArch64: MRC reads of PMEVTYPER<\n> and PMXEVTYPER.

<table>
<thead>
<tr>
<th>PMEVTYPERn_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The operations listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTen == 1, then, unless the read generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MRS reads at EL1 and EL0 using AArch64 of PMEVTYPER&lt;\n&gt;_EL0 and PMXEVTYPER_EL0 are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MRC reads at EL0 using AArch32 of PMEVTYPER&lt;\n&gt; and PMXEVTYPER are trapped to EL2 and reported with EC syndrome value 0x03, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, then, regardless of the value of this bit, for each value n in the range UInt(MDCR_EL2.HPMN) to 30:

• If event counter n is not implemented, the following are UNDEFINED:
  ◦ In AArch64 state, a read of PMEVTYPER<\n>_EL0, or a read of PMXEVTYPER_EL0 when PMSELR_EL0 SEl == n.
  ◦ In AArch32 state, a read of PMEVTYPER<\n>, or a read of PMXEVTYPER when PMSELR.SEl == n.

• If event counter n is implemented, the following generate a Trap exception to EL2:
  ◦ At EL0 or EL1 using AArch64, a read of PMEVTYPER<\n>_EL0, or a read of PMXEVTYPER_EL0 when PMSELR_EL0.SEl == n, reported with EC syndrome value 0x18.
  ◦ At EL0 using AArch32, a read of PMEVTYPER<\n>, or a read of PMXEVTYPER when PMSELR.SEl == n, reported with EC syndrome value 0x03.

See also HDFGRTR_EL2.PMCCFILTR_EL0.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMEVCNTRn_EL0, bit [12]

When PMUv3 is implemented:

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

• At EL1 and EL0 using AArch64: MRS reads of PMEVCNTR<\n>_EL0 and PMXEVCNTR_EL0.

• At EL0 using Arch32 when EL1 is using AArch64: MRC reads of PMEVCNTR<\n> and PMXEVCNTR.
The operations listed above are not affected by this bit.

If EL2 is implemented and enabled in the current Security state, `HCR_EL2` (E2H,TE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or `SCR_EL3` FGTEn == 1, then, unless the read generates a higher priority exception:

- **MRS** reads at EL1 and EL0 using AArch64 of `PMEVCNTR<n>_EL0` and `PMXEVCNTR_EL0` are trapped to EL2 and reported with EC syndrome value \(0x18\).
- **MRC** reads at EL0 using AArch32 of `PMEVCNTR<n>` and `PMXEVCNTR` are trapped to EL2 and reported with EC syndrome value \(0x03\), unless the read generates a higher priority exception.

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, then, regardless of the value of this bit, for each value \(n\) in the range UInt(MDCR_EL2.HPMN) to 30:

- If event counter \(n\) is not implemented, the following are **UNDEFINED**:
  - In AArch64 state, a read of `PMEVCNTR<n>_EL0`, or a read of `PMXEVCNTR_EL0` when `PMSELR_EL0`.SEL == \(n\).
  - In AArch32 state, a read of `PMEVCNTR<n>`, or a read of `PMXEVCNTR` when `PMSELR`.SEL == \(n\).

- If event counter \(n\) is implemented, the following generate a Trap exception to EL2:
  - At EL0 or EL1 using AArch64, a read of `PMEVCNTR<n>_EL0`, or a read of `PMXEVCNTR_EL0` when `PMSELR_EL0`.SEL == \(n\), reported with EC syndrome value \(0x18\).
  - At EL0 using AArch32, a read of `PMEVCNTR<n>`, or a read of `PMXEVCNTR` when `PMSELR`.SEL == \(n\), reported with EC syndrome value \(0x03\).

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### **OSDLR_EL1, bit [11]**

**When ARMv8.0-DoubleLock is implemented:**

Trap **MRS** reads of **OSDLR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>OSDLR_EL1</strong></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>OSDLR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3</code> FGTEn == 1, MRS reads of <strong>OSDLR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value (0x18), unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.
### OSECCR_EL1, bit [10]

Trap MRS reads of **OSECCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>OSECCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>OSECCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>OSECCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### OSLSR_EL1, bit [9]

Trap MRS reads of **OSLSR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>OSLSR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>OSLSR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>OSLSR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### Bit [8]

Reserved, RES0.

### DBGPRCR_EL1, bit [7]

Trap MRS reads of **DBGPRCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DBGPRCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>DBGPRCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>DBGPRCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### DBGAUTHSTATUS_EL1, bit [6]

Trap MRS reads of **DBGAUTHSTATUS_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DBGAUTHSTATUS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>DBGAUTHSTATUS_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>DBGAUTHSTATUS_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

**DBGCLAIM, bit [5]**

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- **DBGCLAIMCLR_EL1**
- **DBGCLAIMSET_EL1**

<table>
<thead>
<tr>
<th>DBGCLAIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**MDSCR_EL1, bit [4]**

Trap MRS reads of **MDSCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>MDSCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>MDSCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>MDSCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**DBGWVRn_EL1, bit [3]**

Trap MRS reads of **DBGWVR<n>_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DBGWVRn_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>DBGWVR&lt;n&gt;_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>DBGWVR&lt;n&gt;_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If watchpoint n is not implemented, a read of **DBGWVR<n>_EL1** is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**DBGWCRn_EL1, bit [2]**

Trap MRS reads of **DBGWCR<n>_EL1** at EL1 using AArch64 to EL2.
If watchpoint n is not implemented, a read of DBGWCR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**DBGVRn_EL1, bit [1]**

Trap MRS reads of DBGVR<n>_EL1 at EL1 using AArch64 to EL2.

If breakpoint n is not implemented, a read of DBGVR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**DBGCRn_EL1, bit [0]**

Trap MRS reads of DBGCR<n>_EL1 at EL1 using AArch64 to EL2.

If breakpoint n is not implemented, a read of DBGCR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**Accessing the HDFGRTR_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, HDFGRTR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1D0];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGRTR_EL2 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    HDFGRTR_EL2 = X[t];

MSR HDFGRTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1D0] = X[t];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGRTR_EL2 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    HDFGRTR_EL2 = X[t];
HDFGWTR_EL2, Hypervisor Debug Fine-Grained Write Trap Register

The HDFGWTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MSR and MCR writes of debug, trace, PMU, and Statistical Profiling System registers.

**Configuration**

This register is present only when ARMv8.6-FGT is implemented. Otherwise, direct accesses to HDFGWTR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HDFGWTR_EL2 is a 64-bit register.

**Field descriptions**

The HDFGWTR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Description</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td><strong>PMUSERENR_EL0</strong>, bit [57]</td>
<td>0</td>
<td>MSR writes of <strong>PMUSERENR_EL0</strong> at EL1 using AArch64 to EL2.</td>
</tr>
<tr>
<td>61</td>
<td><strong>PMICR_EL1</strong>, bit [56]</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td><strong>PMSCR_EL1</strong>, bit [55]</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td><strong>PMSEVFR_EL1</strong>, bit [54]</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td><strong>PMSEVR_EL1</strong>, bit [53]</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td><strong>PMSCR_EL1</strong>, bit [52]</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td><strong>PMSEVFR_EL1</strong>, bit [51]</td>
<td>0</td>
<td>Reserved</td>
</tr>
<tr>
<td>55</td>
<td><strong>PMSEVR_EL1</strong>, bit [50]</td>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:58]**

Reserved, RES0.

**PMUSERENR_EL0, bit [57]**

When PMUv3 is implemented:

- Trap MSR writes of **PMUSERENR_EL0** at EL1 using AArch64 to EL2.

- **PMUSERENR_EL0**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMUSERENR_EL0</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MSR writes of <strong>PMUSERENR_EL0</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
TRBTRG_EL1, bit [56]

When TRBE is implemented:

Trap MSR writes of TRBTRG_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBTRG_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRBTRG_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRBTRG_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBSR_EL1, bit [55]

When TRBE is implemented:

Trap MSR writes of TRBSR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBSR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRBSR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBPTR_EL1, bit [54]

When TRBE is implemented:

Trap MSR writes of TRBPTR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBPTR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRBPTR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
TRBMAR_EL1, bit [53]

When TRBE is implemented:

Trap MSR writes of TRBMAR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBMAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRBMAR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBLIMITR_EL1, bit [52]

When TRBE is implemented:

Trap MSR writes of TRBLIMITR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBLIMITR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRBLIMITR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

TRBBASER_EL1, bit [50]

When TRBE is implemented:

Trap MSR writes of TRBBASER_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRBBASER_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRBBASER_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRBBASER_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

HDFGWTR_EL2, Hypervisor Debug Fine-Grained Write Trap Register
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TRFCR_EL1, bit [49]**

*When ARMv8.4-Trace is implemented:*

Trap MSR writes of **TRFCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRFCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>TRFCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>TRFCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TRCVICTLR, bit [48]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MSR writes of **TRCVICTLR** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCVICTLR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>TRCVICTLR</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>TRCVICTLR</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**Bit [47]**

Reserved, RES0.

**TRCSSCSRn, bit [46]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MSR writes of **TRCSSCSR<n>** at EL1 using AArch64 to EL2.
<table>
<thead>
<tr>
<th>TRCSSCSR&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCSSCSR&lt;n&gt; are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRCSSCSR&lt;n&gt; at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If Single-shot Comparator n is not implemented, a write of TRCSSCSR<n> is **UNDEFINED**.

This bit is **RES0** if TRCSSCSR<n> are not implemented.

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCSEQSTR, bit [45]**

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of TRCSEQSTR at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCSEQSTR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCSEQSTR are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRCSEQSTR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if TRCSEQSTR is not implemented.

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCPRGCTLRL, bit [44]**

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of TRCPRGCTLRL at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCPRGCTLRL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCPRGCTLRL are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRCPRGCTLRL at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.
Bit [43]

Reserved, RES0.

TRCOSLAR, bit [42]

When the Trace Extension is implemented, System register access to the PE Trace Unit registers is implemented and ETMv4 is implemented:

Trap MSR writes of TRCOSLAR at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCOSLAR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCOSLAR are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRCOSLAR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCIMSPECn, bit [41]

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of TRCIMSPEC<n> at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCIMSPECn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCIMSPEC&lt;n&gt; are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRCIMSPEC&lt;n&gt; at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

TRCIMSPEC<1-7> are optional. If TRCIMSPEC<n> is not implemented, a write of TRCIMSPEC<n> is undefined.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCID, bit [40]

When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TRCDEVARCH.
- TRCDEVID.
- TRCIDR<n>.
TRCID | Meaning
--- | ---
0b0 | MSR writes of the System registers listed above are not affected by this bit.
0b1 | If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or \texttt{SCR\_EL3.FGTEn} == 1, MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

TRCCNTVRn, bit [37]

\textbf{When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:}

Trap MSR writes of TRCCNTVR<\texttt{n}> at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCCNTVRn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCCNTVR&lt;\texttt{n}&gt; are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or \texttt{SCR_EL3.FGTEn} == 1, MSR writes of TRCCNTVR&lt;\texttt{n}&gt; at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If Counter n is not implemented, a write of TRCCNTVR<\texttt{n}> is UNDEFINED.

This bit is RES0 if TRCCNTVR<\texttt{n}> are not implemented.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCCLAIM, bit [36]

\textbf{When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:}

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TRCCLAIMCLR.
- TRCCLAIMSET.

<table>
<thead>
<tr>
<th>TRCCLAIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or \texttt{SCR_EL3.FGTEn} == 1, MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**TRCAUXCTLR, bit [35]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MSR writes of TRCAUXCTLR at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TRCAUXCTLR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TRCAUXCTLR are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of TRCAUXCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bit [34]**

Reserved, RES0.

**TRC, bit [33]**

*When the Trace Extension is implemented and System register access to the PE Trace Unit registers is implemented:*

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- TRCACATR<n>.
- TRACACVR<n>.
- TRCCBCTRLR.
- TRCCCTCTRLR.
- TRCCIDCCTRLR<n>.
- TRCCIDCVR<n>.
- TRCCNTCTLR<n>.
- TRCCNTLDRV<n>.
- TRCCONFIGR.
- TRCEVENTCTRL0R.
- TRCEVENTCTRL1R.
- TRCEVENTCTRL2R.
- TRCEXITINSELR<n>, if ETE is implemented.
- TRCEXITINSELR, if ETMv4 is implemented.
- TRCRSCTRLR<n>.
- TRCRSRT, if ETE is implemented.
- TRCSEQEV< n >.
- TRCSEQRSR< n >.
- TRCSSCCCR<n>.
- TRCSSPCICR<n>.
- TRCSSPCICR.< n >.
- TRCSTALLCTRLR.
- TRCSYNCR.
- TRCTRACTGR.
- TRCTSTCTRLR.
- TRCVIIECTRLR.
- TRCVIPCSSCTRLR.
- TRCVIIPCSSCTRLR.
- TRCVVMIDCCTRLR<n>.
• TRCVMIDCVR<n>.

<table>
<thead>
<tr>
<th>TRC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

A write of an unimplemented register is **undefined**.

**TRCEXTINSELR<n>** and **TRCRSR** are only implemented if ETE is implemented.

**TRCEXTINSELR** is only implemented if ETE is not implemented and ETMv4 is implemented.

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, **RES0**.

**PMSLATFR_EL1, bit [32]**

**When SPE is implemented:**

Trap MSR writes of **PMSLATFR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSLATFR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMSLATFR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of <strong>PMSLATFR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, **RES0**.

**PMSIRR_EL1, bit [31]**

**When SPE is implemented:**

Trap MSR writes of **PMSIRR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSIRR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMSIRR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of <strong>PMSIRR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:

Reserved, RES0.

**Bit [30]**

Reserved, RES0.

**PMSICR_EL1, bit [29]**

**When SPE is implemented:**

Trap MSR writes of PMSICR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSICR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of PMSICR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of PMSICR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSFCR_EL1, bit [28]**

**When SPE is implemented:**

Trap MSR writes of PMSFCR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSFCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of PMSFCR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of PMSFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSEVFR_EL1, bit [27]**

**When SPE is implemented:**

Trap MSR writes of PMSEVFR_EL1 at EL1 using AArch64 to EL2.
<table>
<thead>
<tr>
<th>PMSEVFR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMSEVFR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTen == 1, MSR writes of <strong>PMSEVFR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMSCR_EL1**, bit [26]

**When SPE is implemented:**

Trap MSR writes of **PMSCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMSCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTen == 1, MSR writes of <strong>PMSCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMBSR_EL1**, bit [25]

**When SPE is implemented:**

Trap MSR writes of **PMBSR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMBSR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMBSR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTen == 1, MSR writes of <strong>PMBSR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMBPTR_EL1**, bit [24]

**When SPE is implemented:**

Trap MSR writes of **PMBPTR_EL1** at EL1 using AArch64 to EL2.
### PMBPTR_EL1

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### PMBLIMITR_EL1, bit [23]

**When SPE is implemented:**

Trap MSR writes of PMBLIMITR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMBLIMITR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of PMBLIMITR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of PMBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### Bit [22]

Reserved, RES0.

### PMCR_EL0, bit [21]

**When PMUv3 is implemented:**

Trap MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCR at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMCR_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCR at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1      | If EL2 is implemented and enabled in the current Security state, HCR_EL2.(E2H,TGE) != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:  
  * MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  * MCR writes of PMCR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03.  |
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSWINC_EL0, bit [20]**

*When PMUv3 is implemented:*

Trap MSR writes of **PMSWINC_EL0** at EL1 and EL0 using AArch64 and MCR writes of **PMSWINC** at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSWINC_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMSWINC_EL0</strong> at EL1 and EL0 using AArch64 and MCR writes of <strong>PMSWINC</strong> at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1         | If EL2 is implemented and enabled in the current Security state, **HCR_EL2**, {E2H, TGE} \(!\in\{1,1\}\), EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3**, FGTEn \(!\in\{1\}\), then, unless the write generates a higher priority exception:
  * MSR writes of **PMSWINC_EL0** at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  * MCR writes of **PMSWINC** at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMSELR_EL0, bit [19]**

*When PMUv3 is implemented:*

Trap MSR writes of **PMSELR_EL0** at EL1 and EL0 using AArch64 and MCR writes of **PMSELR** at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PMSELR_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>PMSELR_EL0</strong> at EL1 and EL0 using AArch64 and MCR writes of <strong>PMSELR</strong> at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1        | If EL2 is implemented and enabled in the current Security state, **HCR_EL2**, {E2H, TGE} \(!\in\{1,1\}\), EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3**, FGTEn \(!\in\{1\}\), then, unless the write generates a higher priority exception:
  * MSR writes of **PMSELR_EL0** at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  * MCR writes of **PMSELR** at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
PMOV5, bit [18]

When PMUv3 is implemented:

Trap MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of $\text{PMOVSCLR}_{\text{EL}0}$ and $\text{PMOVSSET}_{\text{EL}0}$.
- At EL0 using Arch32 when EL1 is using AArch64: MCR writes of $\text{PMOVSR}$ and $\text{PMOVSSET}$.

<table>
<thead>
<tr>
<th>PMOV5</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The operations listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, $\text{HCR}<em>{\text{EL2}}.{E2H,TGE} != {1,1}$, EL1 is using AArch64, and either EL3 is not implemented or $\text{SCR}</em>{\text{EL3}}.\text{FGTEn} == 1$, then, unless the write generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>- MSR writes at EL1 and EL0 using AArch64 of $\text{PMOVSCLR}<em>{\text{EL}0}$ and $\text{PMOVSSET}</em>{\text{EL}0}$ are trapped to EL2 and reported with EC syndrome value $0x18$.</td>
</tr>
<tr>
<td></td>
<td>- MCR writes at EL0 using AArch32 of $\text{PMOVSR}$ and $\text{PMOVSSET}$ are trapped to EL2 and reported with EC syndrome value $0x03$, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMINTEN, bit [17]

When PMUv3 is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- $\text{PMINTENCLR}_{\text{EL1}}$.
- $\text{PMINTENSET}_{\text{EL1}}$.

<table>
<thead>
<tr>
<th>PMINTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or $\text{SCR}_{\text{EL3}}.\text{FGTEn} == 1$, MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value $0x18$, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCNTEN, bit [16]

When PMUv3 is implemented:

Trap MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:
• At EL1 and EL0 using AArch64: MSR writes of \texttt{PMCNTENCLR\_EL0} and \texttt{PMCNTENSET\_EL0}.

• At EL0 using Arch32 when EL1 is using AArch64: MCR writes of \texttt{PMCNTENCLR} and \texttt{PMCNTENSET}.

<table>
<thead>
<tr>
<th>\texttt{PMC_TEN}</th>
<th>\textit{Meaning}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The operations listed above are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1              | If EL2 is implemented and enabled in the current Security state, \texttt{HCR\_EL2.E2H\_TGE} \(!=\) \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or \texttt{SCR\_EL3.FGTEn} \(!=\) 1, then, unless the write generates a higher priority exception:
  • MSR writes at EL1 and EL0 using AArch64 of \texttt{PMCNTENCLR\_EL0} and \texttt{PMCNTENSET\_EL0} are trapped to EL2 and reported with EC syndrome value \texttt{0x18}.
  • MCR writes at EL0 using AArch32 of \texttt{PMCNTENCLR} and \texttt{PMCNTENSET} are trapped to EL2 and reported with EC syndrome value \texttt{0x03}, unless the write generates a higher priority exception. |

In a system where the PE resets into EL2, this field resets to 0.

\textit{Otherwise:}

Reserved, \texttt{RES0}.

\textbf{PMCCNTR\_EL0, bit [15]}

\textit{When PMUv3 is implemented:}

Trap MSR writes of \texttt{PMCCNTR\_EL0} at EL1 and EL0 using AArch64 and MCR and MCRR writes of \texttt{PMCCNTR} at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>\texttt{PMCCNTR_EL0}</th>
<th>\textit{Meaning}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of \texttt{PMCCNTR_EL0} at EL1 and EL0 using AArch64 and MCR and MCRR writes of \texttt{PMCCNTR} at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1                 | If EL2 is implemented and enabled in the current Security state, \texttt{HCR\_EL2.E2H\_TGE} \(!=\) \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or \texttt{SCR\_EL3.FGTEn} \(!=\) 1, then, unless the write generates a higher priority exception:
  • MSR writes of \texttt{PMCCNTR\_EL0} at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value \texttt{0x18}.
  • MCR and MCRR writes of \texttt{PMCCNTR} at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value \texttt{0x03} (for MCR) or \texttt{0x04} (for MCRR). |

In a system where the PE resets into EL2, this field resets to 0.

\textit{Otherwise:}

Reserved, \texttt{RES0}.

\textbf{PMCCFILTR\_EL0, bit [14]}

\textit{When PMUv3 is implemented:}

Trap MSR writes of \texttt{PMCCFILTR\_EL0} at EL1 and EL0 using AArch64 and MCR writes of \texttt{PMCCFILTR} at EL0 using AArch32 when EL1 is using AArch64 to EL2.
**PMCCFILTR_EL0**

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0</th>
<th>0b1</th>
</tr>
</thead>
</table>
| MSR writes of **PMCCFILTR_EL0** at EL1 and EL0 using AArch64 and MCR writes of **PMCCFILTR** at EL0 using AArch32 are not affected by this bit. | | If EL2 is implemented and enabled in the current Security state, HCR_EL2.E2H,TGE != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:  
  - MSR writes of **PMCCFILTR_EL0** at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  - MCR writes of **PMCCFILTR_EL0** at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

**PMCCFILTR_EL0** can also be accessed in AArch64 state using **PMXEVTYPER_EL0** when PMSEL_EL0.SEL == 31, and **PMCCFILTR** can also be accessed in AArch32 state using **PMXEVTYPER** when PMSEL.SEL == 31.

Setting this bit to 1 has no effect on accesses to **PMXEVTYPER_EL0** and **PMXEVTYPER**, regardless of the value of PMSEL_EL0 or PMSEL.

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**PMEVTYPE Rene_EL0, bit [13]**

When PMUv3 is implemented:

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of **PMEVTYPE<n>_EL0** and **PMXEVTYPER_EL0**.
- At EL0 using Arch32 when EL1 is using AArch64: MCR writes of **PMEVTYPE<n>_EL0** and **PMXEVTYPER_EL0**.

<table>
<thead>
<tr>
<th>PMEVTYPE Rene_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The operations listed above are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.E2H,TGE != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:  
  - MSR writes at EL1 and EL0 using AArch64 of **PMEVTYPE<n>_EL0** and **PMXEVTYPER_EL0** are trapped to EL2 and reported with EC syndrome value 0x18.  
  - MCR writes at EL0 using AArch32 of **PMEVTYPE<n>_EL0** and **PMXEVTYPER** are trapped to EL2 and reported with EC syndrome value 0x03, unless the write generates a higher priority exception. |

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, then, regardless of the value of this bit, for each value n in the range UInt(MDCR_EL2.HPMN) to 30:

- If event counter n is not implemented, the following are UNDEFINED:
  - In AArch64 state, a write of **PMEVTYPE<n>_EL0**, or a write of **PMXEVTYPER_EL0** when PMSEL_EL0.SEL == n.
In AArch32 state, a write of `PMEVTYPER<n>`, or a write of `PMXEVTYPER` when `PMSELR.SEL == n`.

- If event counter n is implemented, the following generate a Trap exception to EL2:
  - At EL0 or EL1 using AArch64, a write of `PMEVTYPER<n>_EL0`, or a write of `PMXEVTYPER_EL0` when `PMSELR_EL0.SEL == n`, reported with EC syndrome value 0x18.
  - At EL0 using AArch32, a write of `PMEVTYPER<n>`, or a write of `PMXEVTYPER` when `PMSELR.SEL == n`, reported with EC syndrome value 0x03.

See also HDFGWTR_EL2.PMCCFILTER_EL0.

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**PMEVCNTRn_EL0, bit [12]**

When PMUv3 is implemented:

Traps MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of `PMEVCNTR<n>_EL0` and `PMXEVCNTR_EL0`.
- At EL0 using Arch32 when EL1 is using AArch64: MCR writes of `PMEVCNTR<n>` and `PMXEVCNTR`.

<table>
<thead>
<tr>
<th>PMEVCNTRn_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The operations listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2 {E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• MSR writes at EL1 and EL0 using AArch64 of <code>PMEVCNTR&lt;n&gt;_EL0</code> and <code>PMXEVCNTR_EL0</code> are trapped to EL2 and reported with EC syndrome value 0x18.</td>
</tr>
<tr>
<td></td>
<td>• MCR writes at EL0 using AArch32 of <code>PMEVCNTR&lt;n&gt;</code> and <code>PMXEVCNTR</code> are trapped to EL2 and reported with EC syndrome value 0x03, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, then, regardless of the value of this bit, for each value n in the range UInt(MDCR_EL2.HPMN) to 30:

- If event counter n is not implemented, the following are UNDEFINED:
  - In AArch64 state, a write of `PMEVCNTR<n>_EL0`, or a write of `PMXEVCNTR_EL0` when `PMSELR_EL0.SEL == n`.
  - In AArch32 state, a write of `PMEVCNTR<n>`, or a write of `PMXEVCNTR` when `PMSELR.SEL == n`.

- If event counter n is implemented, the following generate a Trap exception to EL2:
  - At EL0 or EL1 using AArch64, a write of `PMEVCNTR<n>_EL0`, or a write of `PMXEVCNTR_EL0` when `PMSELR_EL0.SEL == n`, reported with EC syndrome value 0x18.
  - At EL0 using AArch32, a write of `PMEVCNTR<n>`, or a write of `PMXEVCNTR` when `PMSELR.SEL == n`, reported with EC syndrome value 0x03.

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:

Reserved, RES\textsubscript{0}.

**OSDLR\_EL1, bit [11]**

*When ARMv8.0-DoubleLock is implemented:* 

Trap MSR writes of \texttt{OSDLR\_EL1} at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>OSDLR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of \texttt{OSDLR_EL1} are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn (=) 1, MSR writes of \texttt{OSDLR_EL1} at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES\textsubscript{0}.

**OSECCR\_EL1, bit [10]**

Trap MSR writes of \texttt{OSECCR\_EL1} at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>OSECCR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of \texttt{OSECCR_EL1} are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn (=) 1, MSR writes of \texttt{OSECCR_EL1} at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Bit [9]**

Reserved, RES\textsubscript{0}.

**OSLAR\_EL1, bit [8]**

Trap MSR writes of \texttt{OSLAR\_EL1} at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>OSLAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of \texttt{OSLAR_EL1} are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn (=) 1, MSR writes of \texttt{OSLAR_EL1} at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**DBGPRCR\_EL1, bit [7]**

Trap MSR writes of \texttt{DBGPRCR\_EL1} at EL1 using AArch64 to EL2.
## DBGPRCR_EL1

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>DBGPRCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>DBGPRCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### Bit [6]

Reserved, RES0.

### DBGCLAIM, bit [5]

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- **DBGCLAIMCLR_EL1**
- **DBGCLAIMSET_EL1**

### MDSCR_EL1, bit [4]

Trap MSR writes of **MDSCR_EL1** at EL1 using AArch64 to EL2.

### DBGWVRn_EL1, bit [3]

Trap MSR writes of **DBGWVR_<n>_EL1** at EL1 using AArch64 to EL2.
If watchpoint n is not implemented, a write of DBGWVR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**DBGWCRn_EL1, bit [2]**

Trap MSR writes of DBGWCR<n>_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DBGWCRn_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of DBGWCR&lt;n&gt;_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of DBGWCR&lt;n&gt;_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If watchpoint n is not implemented, a write of DBGWCR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**DBGVBRn_EL1, bit [1]**

Trap MSR writes of DBGVBR<n>_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DBGVBRn_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of DBGVBR&lt;n&gt;_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of DBGVBR&lt;n&gt;_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If breakpoint n is not implemented, a write of DBGVBR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**DBGBCRn_EL1, bit [0]**

Trap MSR writes of DBGBCR<n>_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DBGBCRn_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of DBGBCR&lt;n&gt;_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of DBGBCR&lt;n&gt;_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

If breakpoint n is not implemented, a write of DBGBCR<n>_EL1 is UNDEFINED.

In a system where the PE resets into EL2, this field resets to 0.

**Accessing the HDFGWTR_EL2**

Accesses to this register use the following encodings:
MRS <Xt>, HDFGWTR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x1D8];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    HDFGWTR_EL2 = <Xt>;
elsif PSTATE.EL == EL3 then
  HDFGWTR_EL2 = <Xt>;

MSR HDFGWTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x1D8] = <Xt>;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    HDFGWTR_EL2 = <Xt>;
elsif PSTATE.EL == EL3 then
  HDFGWTR_EL2 = <Xt>;

The HFGITR_EL2 characteristics are:

**Purpose**

Provides controls for traps of execution of System instructions.

**Configuration**

This register is present only when ARMv8.6-FGT is implemented. Otherwise, direct accesses to HFGITR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HFGITR_EL2 is a 64-bit register.

**Field descriptions**

The HFGITR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>TLIBVAEE1IS</td>
<td>TLIBIASIDE1IS</td>
<td>TLIBVAE1IS</td>
<td>TLIBVMALLE1IS</td>
<td>TLIBIRAEE1OS</td>
<td>TLIBIRAEE1OS</td>
<td>TLIBIRAEE1OS</td>
</tr>
</tbody>
</table>

**Bits [63:55]**

Reserved, RES0.

**DCCVAC, bit [54]**

Trap execution of multiple System instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC CGDVCAC**, if ARMv8.5-MemTag is implemented.
- **DC CGVAC**, if ARMv8.5-MemTag is implemented.
- **DC CVAC**.

<table>
<thead>
<tr>
<th>DCCVAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2 {E2H,TGE} != {1,1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution at EL1 and EL0 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
**SVC_EL1, bit [53]**

Trap execution of SVC at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>SVC_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of SVC is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of SVC at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x15, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**SVC_EL0, bit [52]**

Trap execution of SVC at EL0 using AArch64 and execution of SVC at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>SVC_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of SVC at EL0 using AArch64 and execution of SVC at EL0 using AArch32 is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the instruction generates a higher priority exception:</td>
</tr>
<tr>
<td></td>
<td>• Execution of SVC at EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x15.</td>
</tr>
<tr>
<td></td>
<td>• Execution of SVC at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x11.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**ERET, bit [51]**

Trap execution of multiple System instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 System instructions to EL2:

- ERET.
- ERETTAA, if ARMv8.3-PAuth is implemented.
- ERETTAB, if ARMv8.3-PAuth is implemented.

<table>
<thead>
<tr>
<th>ERET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution at EL1 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x1A, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**CPPRCTX, bit [50]**

*When ARMv8.0-PredInv is implemented:*

Trap execution of CPP RCTX at EL1 and EL0 using AArch64 and execution of CPPRCTX at EL0 using AArch32 when EL1 is using AArch64 to EL2.
### CPPRCTX

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
</tbody>
</table>
| 0b1     | If EL2 is implemented and enabled in the current Security state, **HCR_EL2**.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3**.FGTEn == 1, then, unless the instruction generates a higher priority exception:  
  - Execution of **CPP RCTX** at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18.  
  - Execution of **CPPRCTX** at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### DVPRCTX, bit [49]

When ARMv8.0-PredInv is implemented:

Trap execution of **DVP RCTX** at EL1 and EL0 using AArch64 and execution of **DVPRCTX** at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
</tbody>
</table>
| 0b1     | If EL2 is implemented and enabled in the current Security state, **HCR_EL2**.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3**.FGTEn == 1, then, unless the instruction generates a higher priority exception:  
  - Execution of **DVP RCTX** at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18.  
  - Execution of **DVPRCTX** at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### CFPRCTX, bit [48]

When ARMv8.0-PredInv is implemented:

Trap execution of **CFP RCTX** at EL1 and EL0 using AArch64 and execution of **CFPRCTX** at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
</tbody>
</table>
| 0b1     | If EL2 is implemented and enabled in the current Security state, **HCR_EL2**.\{E2H,TGE\} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3**.FGTEn == 1, then, unless the instruction generates a higher priority exception:  
  - Execution of **CFP RCTX** at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18.  
  - Execution of **CFPRCTX** at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.
### CFPRCTX

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
</tbody>
</table>
| 0b1 | If EL2 is implemented and enabled in the current Security state, **HCR_EL2**.{E2H,TGE} != \{1,1\}, EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3**.FGTEn == 1, then, unless the instruction generates a higher priority exception:  
  - Execution of **CFP RCTX** at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18.  
  - Execution of **CFPRCTX** at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### TLBIVALE1, bit [47]

Trap execution of **TLBI VALE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVALE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VALE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>TLBI VALE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### TLBIVALE1, bit [46]

Trap execution of **TLBI VALE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVALE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VALE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>TLBI VALE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### TLBIVAAE1, bit [45]

Trap execution of **TLBI VAEE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAAE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VAEE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>TLBI VAEE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
In a system where the PE resets into EL2, this field resets to 0.

**TLBIASIDE1, bit [44]**

Trap execution of **TLBI ASIDE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIASIDE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI ASIDE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, execution of <strong>TLBI ASIDE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TLBIVAE1, bit [43]**

Trap execution of **TLBI VAE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VAE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, execution of <strong>TLBI VAE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TLBIVMALLE1, bit [42]**

Trap execution of **TLBI VMALLE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVMALLE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VMALLE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, execution of <strong>TLBI VMALLE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TLBIRVAALE1, bit [41]**

When ARMv8.4-TLBI is implemented:

Trap execution of **TLBI RVAALE1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVAALE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI RVAALE1</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, execution of <strong>TLBI RVAALE1</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:

Reserved, RES0.

**TLBIRVALE1, bit [40]**

When ARMv8.4-TLBI is implemented:

Trap execution of \texttt{TLBI RVALE1} at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVALE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\texttt{b0}</td>
<td>Execution of \texttt{TLBI RVALE1} is not affected by this bit.</td>
</tr>
<tr>
<td>0\texttt{b1}</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or \texttt{SCR_EL3.FGTEn == 1}, execution of \texttt{TLBI RVALE1} at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value \texttt{0x18}, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to \texttt{0}.

Otherwise:

Reserved, RES0.

**TLBIRVAEE1, bit [39]**

When ARMv8.4-TLBI is implemented:

Trap execution of \texttt{TLBI RVAAE1} at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVAEE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\texttt{b0}</td>
<td>Execution of \texttt{TLBI RVAAE1} is not affected by this bit.</td>
</tr>
<tr>
<td>0\texttt{b1}</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or \texttt{SCR_EL3.FGTEn == 1}, execution of \texttt{TLBI RVAAE1} at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value \texttt{0x18}, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to \texttt{0}.

Otherwise:

Reserved, RES0.

**TLBIRVAE1, bit [38]**

When ARMv8.4-TLBI is implemented:

Trap execution of \texttt{TLBI RVAE1} at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVAE1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\texttt{b0}</td>
<td>Execution of \texttt{TLBI RVAE1} is not affected by this bit.</td>
</tr>
<tr>
<td>0\texttt{b1}</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or \texttt{SCR_EL3.FGTEn == 1}, execution of \texttt{TLBI RVAE1} at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value \texttt{0x18}, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to \texttt{0}.
Reserved, RES0.

**TLBIRVALE1IS, bit [37]**

When ARMv8.4-TLBI is implemented:

Trap execution of TLBI RVALE1IS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVALE1IS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of TLBI RVALE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of TLBI RVALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Reserved, RES0.

**TLBIRVALE1IS, bit [36]**

When ARMv8.4-TLBI is implemented:

Trap execution of TLBI RVALE1IS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVALE1IS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of TLBI RVALE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of TLBI RVALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Reserved, RES0.

**TLBIRVAEE1IS, bit [35]**

When ARMv8.4-TLBI is implemented:

Trap execution of TLBI RVAAE1IS at EL1 using AArch64 to EL2.
### TLBIRVAE11S

<table>
<thead>
<tr>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 Execution of <strong>TLBI RVAE11S</strong> is not affected by this bit.</td>
<td></td>
</tr>
<tr>
<td>0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of <strong>TLBI RVAE11S</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
<td></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### TLBIVAALE11S, bit [34]

When ARMv8.4-TLBi is implemented:

Trap execution of **TLBI RVAE11S** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 Execution of <strong>TLBI RVAE11S</strong> is not affected by this bit.</td>
<td></td>
</tr>
<tr>
<td>0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of <strong>TLBI RVAE11S</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
<td></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### TLBIVALE11S, bit [33]

Trap execution of **TLBI VAALE11S** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 Execution of <strong>TLBI VALE11S</strong> is not affected by this bit.</td>
<td></td>
</tr>
<tr>
<td>0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of <strong>TLBI VALE11S</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
<td></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### TLBIVALE11S, bit [32]

Trap execution of **TLBI VALE11S** at EL1 using AArch64 to EL2.
TLBIVALE1IS, bit [31]

Trap execution of TLBI VALE1IS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVALE1IS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of TLBI VALE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3_FGTEn == 1, execution of TLBI VALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TLBIVAAE1IS, bit [30]

Trap execution of TLBI VAEE1IS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAAE1IS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of TLBI VAEE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3_FGTEn == 1, execution of TLBI VAEE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TLBIASIDE1IS, bit [29]

Trap execution of TLBI ASIDE1IS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIASIDE1IS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of TLBI ASIDE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3_FGTEn == 1, execution of TLBI ASIDE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TLBIVAE1IS, bit [28]

Trap execution of TLBI VAEE1IS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAE1IS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of TLBI VAE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3_FGTEn == 1, execution of TLBI VAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TLBIVMALLE1IS, bit [27]

Trap execution of TLBI VMALLE1IS at EL1 using AArch64 to EL2.
TLBIVMALE1IS

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execution of TLBI VMALLE1IS is not affected by this bit.</td>
</tr>
<tr>
<td>1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTe == 1, execution of TLBI VMALLE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TLBIRVALE1OS, bit [27]

When ARMv8.4-TLB is implemented:

Trap execution of TLBI RVALE1OS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execution of TLBI RVALE1OS is not affected by this bit.</td>
</tr>
<tr>
<td>1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTe == 1, execution of TLBI RVALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVALE1OS, bit [26]

When ARMv8.4-TLB is implemented:

Trap execution of TLBI RVALE1OS at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Execution of TLBI RVALE1OS is not affected by this bit.</td>
</tr>
<tr>
<td>1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTe == 1, execution of TLBI RVALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAAE1OS, bit [25]

When ARMv8.4-TLB is implemented:

Trap execution of TLBI RVAE1OS at EL1 using AArch64 to EL2.
<table>
<thead>
<tr>
<th>TLBIRVAEE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <code>TLBI RVAE1EOS</code> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution of <code>TLBI RVAE1EOS</code> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value <code>0x18</code>, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TLBIRVAE1OS, bit [24]**

_When ARMv8.4-TLBI is implemented:_

Trap execution of `TLBI RVAE1EOS` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIRVAE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <code>TLBI RVAE1EOS</code> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution of <code>TLBI RVAE1EOS</code> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value <code>0x18</code>, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**TLBIVAALE1OS, bit [23]**

_When ARMv8.4-TLBI is implemented:_

Trap execution of `TLBI VAALE1EOS` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAALE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <code>TLBI VAALE1EOS</code> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution of <code>TLBI VAALE1EOS</code> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value <code>0x18</code>, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
When ARMv8.4-TLBI is implemented:

Trap execution of **TLBI VALE1OS** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVALE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VALE1OS</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>TLBI VALE1OS</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

When ARMv8.4-TLBI is implemented:

Trap execution of **TLBI VAAE1OS** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAAE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI VAAE1OS</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>TLBI VAAE1OS</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

When ARMv8.4-TLBI is implemented:

Trap execution of **TLBI ASIDE1OS** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIASIDE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBI ASIDE1OS</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>TLBI ASIDE1OS</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.
When ARMv8.4-TLB1 is implemented:

Trap execution of **TLBIVAE1OS** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBIVAE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBIVAE1OS</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>TLBIVAE1OS</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

When ARMv8.4-TLB1 is implemented:

Trap execution of **TLBVMALE1OS** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TLBLVMALE1OS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>TLBVMALE1OS</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>TLBVMALE1OS</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

When ARMv8.2-ATS1E1 is implemented:

Trap execution of **ATS1E1WP** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ATS1E1WP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>ATS1E1WP</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>ATS1E1WP</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
ATS1E1RP, bit [16]

When ARMv8.2-ATS1E1 is implemented:

Trap execution of **ATS1E1RP** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ATS1E1RP</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>ATS1E1RP</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>ATS1E1RP</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ATS1E0W, bit [15]

Trap execution of **ATS1E0W** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ATS1E0W</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>ATS1E0W</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>ATS1E0W</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

ATS1E0R, bit [14]

Trap execution of **ATS1E0R** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ATS1E0R</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>ATS1E0R</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>ATS1E0R</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

ATS1E1W, bit [13]

Trap execution of **ATS1E1W** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ATS1E1W</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>ATS1E1W</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, execution of <strong>ATS1E1W</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
ATS1E1R, bit [12]

Trap execution of **ATS1E1R** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ATS1E1R</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>ATS1E1R</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution of <strong>ATS1E1R</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

DCZVA, bit [11]

Trap execution of multiple System instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC GVA**, if ARMv8.5-MemTag is implemented.
- **DC GZVA**, if ARMv8.5-MemTag is implemented.
- **DC ZVA**.

<table>
<thead>
<tr>
<th><strong>DCZVA</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <strong>HCR_EL2.(E2H,TGE) != {1,1}</strong>, and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution at EL1 and EL0 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

DCCIVAC, bit [10]

Trap execution of multiple System instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC CIGDVAC**, if ARMv8.5-MemTag is implemented.
- **DC CIGVAC**, if ARMv8.5-MemTag is implemented.
- **DC CIVAC**.

<table>
<thead>
<tr>
<th><strong>DCCIVAC</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <strong>HCR_EL2.(E2H,TGE) != {1,1}</strong>, and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, execution at EL1 and EL0 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

DCCVADP, bit [9]

**When ARMv8.2-DCCVADP is implemented:**

Trap execution of multiple System instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC CGDVADP**, if ARMv8.5-MemTag is implemented.
- **DC CGVADP**, if ARMv8.5-MemTag is implemented.
- **DC CVADP**.
### DCCVADP

<table>
<thead>
<tr>
<th>Meanings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <code>HCR_EL2.(E2H,TGE) != {1,1}</code> and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution at EL1 and EL0 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### Otherwise:

Reserved, RES0.

### DCCVAP, bit [8]

Trap execution of multiple System instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC CGDVAP**, if ARMv8.5-MemTag is implemented.
- **DC CGVAP**, if ARMv8.5-MemTag is implemented.
- **DC CVAP**.

<table>
<thead>
<tr>
<th>Meanings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <code>HCR_EL2.(E2H,TGE) != {1,1}</code>, and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution at EL1 and EL0 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### DCCVAU, bit [7]

Trap execution of **DC CVAU** at EL1 and EL0 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meanings</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>DC CVAU</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <code>HCR_EL2.(E2H,TGE) != {1,1}</code>, and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution of <strong>DC CVAU</strong> at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### DCCISW, bit [6]

Trap execution of multiple System instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC CIGDSW**, if ARMv8.5-MemTag is implemented.
- **DC CIGSW**, if ARMv8.5-MemTag is implemented.
- **DC CISW**.
### DCCISW

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td><strong>0b1</strong> If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, execution at EL1 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### DCISW, bit [4]

Trap execution of multiple System instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC IGDVAC**, if ARMv8.5-MemTag is implemented.
- **DC IGVAC**, if ARMv8.5-MemTag is implemented.
- **DC IVAC**.

### DCIVAC, bit [3]

Trap execution of multiple System instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 System instructions to EL2:

- **DC IGDVAC**, if ARMv8.5-MemTag is implemented.
- **DC IGVAC**, if ARMv8.5-MemTag is implemented.
- **DC IVAC**.
<table>
<thead>
<tr>
<th>DCIVAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the System instructions listed above is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution at EL1 using AArch64 of any of the System instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**ICIVAU, bit [2]**

Trap execution of **IC IVAU** at EL1 and EL0 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ICIVAU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>IC IVAU</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of <strong>IC IVAU</strong> at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**ICIALLU, bit [1]**

Trap execution of **IC IALLU** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ICIALLU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>IC IALLU</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of <strong>IC IALLU</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**ICIALLUIS, bit [0]**

Trap execution of **IC IALLUIS** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ICIALL UIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of <strong>IC IALLUIS</strong> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, execution of <strong>IC IALLUIS</strong> at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Accessing the HFGITR_EL2**

Accesses to this register use the following encodings:

```
MRS <Xt>, HFGITR_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>

---

**Note:**

The information above is a summary of the document content. For a more detailed understanding, please refer to the original document.
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1C8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            HFGITR_EL2 = X[t];
        elsif PSTATE.EL == EL3 then
            HFGITR_EL2 = X[t];

endif

MSR HFGITR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1C8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            HFGITR_EL2 = X[t];
        elsif PSTATE.EL == EL3 then
            HFGITR_EL2 = X[t];

endif

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
HFGRTR_EL2, Hypervisor Fine-Grained Read Trap Register

The HFGRTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MRS and MRC reads of System registers.

**Configuration**

This register is present only when ARMv8.6-FGT is implemented. Otherwise, direct accesses to HFGRTR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HFGRTR_EL2 is a 64-bit register.

**Field descriptions**

The HFGRTR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCXTNUM_EL0</td>
<td>SCXTNUM_EL1</td>
<td>SCTRL_EL1</td>
<td>REVIDR_EL1</td>
<td>PAR_EL1</td>
<td>MPIDR_EL1</td>
<td>MIDR_EL1</td>
<td>MAIR_EL1</td>
<td>LORSA_EL1</td>
<td>LORN_EL1</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
</tbody>
</table>

**Bits [63:50]**

Reserved, RES0.

**ERXADDR_EL1, bit [49]**

When RAS is implemented:

Trap MRS reads of ERXADDR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXADDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of ERXADDR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
ERXPFGCDN_EL1, bit [48]
When ARMv8.4-RAS is implemented:

Trap MRS reads of **ERXPFGCDN_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ERXPFGCDN_EL1</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>ERXPFGCDN_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>ERXPFGCDN_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFGCTL_EL1, bit [47]
When ARMv8.4-RAS is implemented:

Trap MRS reads of **ERXPFGCTL_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ERXPFGCTL_EL1</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>ERXPFGCTL_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>ERXPFGCTL_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFGF_EL1, bit [46]
When RAS is implemented:

Trap MRS reads of **ERXPFGF_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>ERXPFGF_EL1</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>ERXPFGF_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>ERXPFGF_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:
Reserved, RES0.

**ERXMISC<El1, bit [45]**
When RAS is implemented:

Trap MRS reads of ERXMISC<El1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXMISC&lt;El1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of ERXMISC&lt;El1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of ERXMISC&lt;El1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**ERXSTATUS_EL1, bit [44]**
When RAS is implemented:

Trap MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXSTATUS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of ERXSTATUS_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**ERXCTLR_EL1, bit [43]**
When RAS is implemented:

Trap MRS reads of ERXCTLR_EL1 at EL1 using AArch64 to EL2.
ERXCTLR_EL1, Hypervisor Fine-Grained Read Trap Register

<table>
<thead>
<tr>
<th>ERXCTLR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of ERXCTLR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXFR_EL1, bit [42]

When RAS is implemented:

Trap MRS reads of ERXFR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXFR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of ERXFR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of ERXFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERRSELR_EL1, bit [41]

When RAS is implemented:

Trap MRS reads of ERRSELR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERRSELR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of ERRSELR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
ERRIDR_EL1, bit [40]

When RAS is implemented:

Trap MRS reads of `ERRIDR_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERRIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <code>ERRIDR_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MRS reads of <code>ERRIDR_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value <code>0x18</code>, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When GICv3 is implemented:

Trap MRS reads of `ICC_IGRPEN<n>_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ICC_IGRPENn_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <code>ICC_IGRPEN&lt;n&gt;_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MRS reads of <code>ICC_IGRPEN&lt;n&gt;_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value <code>0x18</code>, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MRS reads of `VBAR_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>VBAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <code>VBAR_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MRS reads of <code>VBAR_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value <code>0x18</code>, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TTBR1_EL1, bit [37]

Trap MRS reads of `TTBR1_EL1` at EL1 using AArch64 to EL2.
HFGRTR_EL2, Hypervisor Fine-Grained Read Trap Register

<table>
<thead>
<tr>
<th>TTBR1_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TTBR1_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TTBR0_EL1, bit [36]**

Trap MRS reads of TTBR0_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TTBR0_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TTBR0_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TPIDR_EL0, bit [35]**

Trap MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>TPIDR_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1       | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception:
  * MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  * MRC reads of TPIDRURW at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**TPIDRRO_EL0, bit [34]**

Trap MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 when EL1 is using AArch64 to EL2.
TPIDRRO_EL0

<table>
<thead>
<tr>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
</tbody>
</table>
| **0b1** | If EL2 is implemented and enabled in the current Security state, **HCR_EL2.\{E2H,TGE\} != \{1,1\}**, EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3.FGTEn == 1**, then, unless the read generates a higher priority exception:
  - MRS reads of **TPIDRRO_EL0** at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  - MRC reads of **TPIDRURO** at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

TPIDR_EL1, bit [33]

**Trap** MRS reads of **TPIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
<tr>
<td><strong>0b1</strong></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

TCR_EL1, bit [32]

**Trap** MRS reads of **TCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
<tr>
<td><strong>0b1</strong></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

SCXTNUM_EL0, bit [31]

When ARMv8.0-CSV2 is implemented:

**Trap** MRS reads of **SCXTNUM_EL0** at EL1 and EL0 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
</tr>
<tr>
<td><strong>0b1</strong></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:
Reserved, RES0.

**SCXTNUM_EL1, bit [30]**

When ARMv8.0-CSV2 is implemented:

Trap MRS reads of `SCXTNUM_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>SCXTNUM_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <code>SCXTNUM_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MRS reads of <code>SCXTNUM_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

**SCTLR_EL1, bit [29]**

Trap MRS reads of `SCTLR_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>SCTLR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <code>SCTLR_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MRS reads of <code>SCTLR_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**REVIDR_EL1, bit [28]**

Trap MRS reads of `REVIDR_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>REVIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <code>REVIDR_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MRS reads of <code>REVIDR_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**PAR_EL1, bit [27]**

Trap MRS reads of `PAR_EL1` at EL1 using AArch64 to EL2.
Meaning

0b0 MRS reads of PAR_EL1 are not affected by this bit.

0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

In a system where the PE resets into EL2, this field resets to 0.

**MPIDR_EL1, bit [26]**

Trap MRS reads of MPIDR_EL1 at EL1 using AArch64 to EL2.

Meaning

0b0 MRS reads of MPIDR_EL1 are not affected by this bit.

0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of MPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

In a system where the PE resets into EL2, this field resets to 0.

**MIDR_EL1, bit [25]**

Trap MRS reads of MIDR_EL1 at EL1 using AArch64 to EL2.

Meaning

0b0 MRS reads of MIDR_EL1 are not affected by this bit.

0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of MIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

In a system where the PE resets into EL2, this field resets to 0.

**MAIR_EL1, bit [24]**

Trap MRS reads of MAIR_EL1 at EL1 using AArch64 to EL2.

Meaning

0b0 MRS reads of MAIR_EL1 are not affected by this bit.

0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.

In a system where the PE resets into EL2, this field resets to 0.

**LORSA_EL1, bit [23]**

When ARMv8.1-LOR is implemented:

Trap MRS reads of LORSA_EL1 at EL1 using AArch64 to EL2.
HFGTR_EL2, Hypervisor Fine-Grained Read Trap Register

<table>
<thead>
<tr>
<th>LORSA_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of LORSA_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**LORN_EL1, bit [22]**

When ARMv8.1-LOR is implemented:

Trap MRS reads of LORN_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LORN_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of LORN_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**LORID_EL1, bit [21]**

When ARMv8.1-LOR is implemented:

Trap MRS reads of LORID_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LORID_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of LORID_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of LORID_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**LOREA_EL1, bit [20]**

When ARMv8.1-LOR is implemented:

Trap MRS reads of LOREA_EL1 at EL1 using AArch64 to EL2.
In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**LORC_EL1, bit [19]**

When ARMv8.1-LOR is implemented:

Trap MRS reads of **LORC_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LORC_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>LORC_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>LOREA_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

**ISR_EL1, bit [18]**

Trap MRS reads of **ISR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ISR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>ISR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>LOREA_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**FAR_EL1, bit [17]**

Trap MRS reads of **FAR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>FAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>FAR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn</strong> == 1, MRS reads of <strong>FAR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
**ESR_EL1, bit [16]**

Trap MRS reads of **ESR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ESR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>ESR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>ESR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**DCZID_EL0, bit [15]**

Trap MRS reads of **DCZID_EL0** at EL1 and EL0 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>DCZID_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>DCZID_EL0</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <strong>HCR_EL2.{E2H,TGE} != {1,1}</strong>, and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>DCZID_EL0</strong> at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**CTR_EL0, bit [14]**

Trap MRS reads of **CTR_EL0** at EL1 and EL0 using AArch64 and MRC reads of **CTR** at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CTR_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>CTR_EL0</strong> at EL1 and EL0 using AArch64 and MRC reads of <strong>CTR</strong> at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1     | If EL2 is implemented and enabled in the current Security state, **HCR_EL2.{E2H,TGE} != {1,1}**, EL1 is using AArch64, and either EL3 is not implemented or **SCR_EL3.FGTEn == 1**, then, unless the read generates a higher priority exception:
  - MRS reads of **CTR_EL0** at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.
  - MRC reads of **CTR** at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.

**CSSELR_EL1, bit [13]**

Trap MRS reads of **CSSELR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CSSELR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>CSSELR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3.FGTEn == 1</strong>, MRS reads of <strong>CSSELR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
**CPACR_EL1, bit [12]**

Trap MRS reads of **CPACR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CPACR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>CPACR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MRS reads of <strong>CPACR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**CONTEXTIDR_EL1, bit [11]**

Trap MRS reads of **CONTEXTIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CONTEXTIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>CONTEXTIDR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MRS reads of <strong>CONTEXTIDR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**CLIDR_EL1, bit [10]**

Trap MRS reads of **CLIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CLIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>CLIDR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MRS reads of <strong>CLIDR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**CCSIDR_EL1, bit [9]**

Trap MRS reads of **CCSIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CCSIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>CCSIDR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MRS reads of <strong>CCSIDR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
APIBKey, bit [8]

When ARMv8.3-PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

<table>
<thead>
<tr>
<th>APIBKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]

When ARMv8.3-PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

<table>
<thead>
<tr>
<th>APIAKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]

When ARMv8.3-PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.
### APGAKey

<table>
<thead>
<tr>
<th>APGAKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### APDBKey, bit [5]

When ARMv8.3-PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDBKeyHi_EL1
- APDBKeyLo_EL1

<table>
<thead>
<tr>
<th>APDBKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

### APDAKey, bit [4]

When ARMv8.3-PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APDAKeyHi_EL1
- APDAKeyLo_EL1

<table>
<thead>
<tr>
<th>APDAKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:

Reserved, RES0.

**AMAIR_EL1, bit [3]**

Trap MRS reads of **AMAIR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMAIR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>AMAIR_EL1</strong> are not affected by this bit. If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>AMAIR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
<tr>
<td>0b1</td>
<td></td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**AIDR_EL1, bit [2]**

Trap MRS reads of **AIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>AIDR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>AIDR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**AFSR1_EL1, bit [1]**

Trap MRS reads of **AFSR1_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AFSR1_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>AFSR1_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>AFSR1_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**AFSR0_EL1, bit [0]**

Trap MRS reads of **AFSR0_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AFSR0_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MRS reads of <strong>AFSR0_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MRS reads of <strong>AFSR0_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Accessing the HFGTR_EL2

Accesses to this register use the following encodings:

MRS <Xt>, HFGTR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1B8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HFGTR_EL2;
elsif PSTATE.EL == EL3 then
    return HFGTR_EL2;

MSR HFGTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1B8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HFGTR_EL2 = X[t];
HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register

The HFGWTR_EL2 characteristics are:

**Purpose**

Provides controls for traps of MSR and MCR writes of System registers.

**Configuration**

This register is present only when ARMv8.6-FGT is implemented. Otherwise, direct accesses to HFGWTR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HFGWTR_EL2 is a 64-bit register.

**Field descriptions**

The HFGWTR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCXTNUM_EL0</td>
<td>SCXTNUM_EL1</td>
<td>SCTRL_EL1</td>
<td>RES0</td>
<td>PAR_EL1</td>
<td>RES0</td>
<td>MAIR_EL1</td>
<td>LORSA_EL1</td>
<td>LORN_EL1</td>
<td>RES0</td>
<td>LOREA_EL1</td>
<td>LORC_EL1</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
</tr>
</tbody>
</table>

**Bits [63:50]**

Reserved, RES0.

**ERXADDR_EL1, bit [49]**

*When RAS is implemented:*

Trap MSR writes of ERXADDR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXADDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of ERXADDR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.
ERXPFGCDN_EL1, bit [48]

When ARMv8.4-RAS is implemented:

Trap MSR writes of ERXPFGCDN_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXPFGCDN_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of ERXPFGCDN_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERXPFGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFGCTL_EL1, bit [47]

When ARMv8.4-RAS is implemented:

Trap MSR writes of ERXPFGCTL_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERXPFGCTL_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of ERXPFGCTL_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERXPFGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [46]

Reserved, RES0.

ERXMISCn_EL1, bit [45]

When RAS is implemented:

Trap MSR writes of ERXMISC<n>_EL1 at EL1 using AArch64 to EL2.
ERXMISCn_EL1

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 MSR writes of ERXMISC&lt;n&gt;_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERXMISC&lt;n&gt;_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When RAS is implemented:

Trap MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 MSR writes of ERXSTATUS_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When RAS is implemented:

Trap MSR writes of ERXCTLR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 MSR writes of ERXCTLR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1 If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.
Bit [42]

Reserved, RES0.

ERRSEL_EL1, bit [41]

When RAS is implemented:

Trap MSR writes of ERRSEL_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ERRSEL_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of ERRSEL_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ERRSEL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [40]

Reserved, RES0.

ICC_IGRPEN<n>_EL1, bit [39]

When GICv3 is implemented:

Trap MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>ICC_IGRPEN&lt;n&gt;_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of ICC_IGRPEN&lt;n&gt;_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of ICC_IGRPEN&lt;n&gt;_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MSR writes of VBAR_EL1 at EL1 using AArch64 to EL2.
## VBAR_EL1

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of VBAR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

## TTBR1_EL1, bit [37]

Trap MSR writes of TTBR1_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TTBR1_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

## TTBR0_EL1, bit [36]

Trap MSR writes of TTBR0_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TTBR0_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

## TPIDR_EL0, bit [35]

Trap MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and MCR writes of TPIDRURW at EL0 using AArch32 when EL1 is using AArch64 to EL2.

<table>
<thead>
<tr>
<th>Bit Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and MCR writes of TPIDRURW at EL0 using AArch32 are not affected by this bit.</td>
</tr>
</tbody>
</table>
| 0b1       | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H,TGE} != {1,1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGEn == 1, then, unless the write generates a higher priority exception:  
  • MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.  
  • MCR writes of TPIDRURW at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

In a system where the PE resets into EL2, this field resets to 0.
**TPIDR_EL0, bit [34]**

Trap MSR writes of **TPIDR_EL0** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>TPIDR_EL0</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>TPIDR_EL0</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>TPIDR_EL0</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TPIDR_EL1, bit [33]**

Trap MSR writes of **TPIDR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>TPIDR_EL1</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>TPIDR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>TPIDR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**TCR_EL1, bit [32]**

Trap MSR writes of **TCR_EL1** at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>TCR_EL1</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>TCR_EL1</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>TCR_EL1</strong> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**SCXTNUM_EL0, bit [31]**

*When ARMv8.0-CSV2 is implemented:*

Trap MSR writes of **SCXTNUM_EL0** at EL1 and EL0 using AArch64 to EL2.

<table>
<thead>
<tr>
<th><strong>SCXTNUM_EL0</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <strong>SCXTNUM_EL0</strong> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, <strong>HCR_EL2</strong>.E2H,TGE != {1,1}, and either EL3 is not implemented or <strong>SCR_EL3</strong>.FGTEn == 1, MSR writes of <strong>SCXTNUM_EL0</strong> at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:
Reserved, RES0.

SCXTNUM_EL1, bit [30]

When ARMv8.0-CSV2 is implemented:

Trap MSR writes of SCXTNUM_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>SCXTNUM_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of SCXTNUM_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:
Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MSR writes of SCTLR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>SCTLR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of SCTLR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of SCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Bit [28]

Reserved, RES0.

PAR_EL1, bit [27]

Trap MSR writes of PAR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>PAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of PAR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Bits [26:25]

Reserved, RES0.

Page 749
MAIR_EL1, bit [24]

Trap MSR writes of `MAIR_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>MAIR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <code>MAIR_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MSR writes of <code>MAIR_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

LORSA_EL1, bit [23]

When ARMv8.1-LOR is implemented:

Trap MSR writes of `LORSA_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LORSA_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <code>LORSA_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MSR writes of <code>LORSA_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORN_EL1, bit [22]

When ARMv8.1-LOR is implemented:

Trap MSR writes of `LORN_EL1` at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LORN_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of <code>LORN_EL1</code> are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or <code>SCR_EL3.FGTEn == 1</code>, MSR writes of <code>LORN_EL1</code> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.
LOREA_EL1, bit [20]

When ARMv8.1-LOR is implemented:

Trap MSR writes of LOREA_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LOREA_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of LOREA_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When ARMv8.1-LOR is implemented:

Trap MSR writes of LORC_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>LORC_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of LORC_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

FAR_EL1, bit [17]

Trap MSR writes of FAR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>FAR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of FAR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

ESR_EL1, bit [16]

Trap MSR writes of ESR_EL1 at EL1 using AArch64 to EL2.
ESR_EL1

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of ESR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Bits [15:14]

Reserved, RES0.

CSSELRL_EL1, bit [13]

Trap MSR writes of CSSELRL_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CSSELRL_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of CSSELRL_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of CSSELRL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

CPACR_EL1, bit [12]

Trap MSR writes of CPACR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CPACR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of CPACR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

CONTEXTIDR_EL1, bit [11]

Trap MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>CONTEXTIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of CONTEXTIDR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGEn == 1, MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Bits [10:9]

Reserved, RES0.
APIBKey, bit [8]

When ARMv8.3-PAuth is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIBKeyHi_EL1.
- APIBKeyLo_EL1.

<table>
<thead>
<tr>
<th>APIBKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]

When ARMv8.3-PAuth is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APIAKeyHi_EL1.
- APIAKeyLo_EL1.

<table>
<thead>
<tr>
<th>APIAKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]

When ARMv8.3-PAuth is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- APGAKeyHi_EL1.
- APGAKeyLo_EL1.
HGWTR_EL2, Hypervisor Fine-Grained Write Trap Register

<table>
<thead>
<tr>
<th>APGAKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or ( \text{SCR}_\text{EL3}.\text{FGTEn} == 1 ), MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**APDBKey, bit [5]**

*When ARMv8.3-PAuth is implemented:*

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- \( \text{APDBKeyHi}_\text{EL1} \).
- \( \text{APDBKeyLo}_\text{EL1} \).

<table>
<thead>
<tr>
<th>APDBKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or ( \text{SCR}_\text{EL3}.\text{FGTEn} == 1 ), MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**APDAKey, bit [4]**

*When ARMv8.3-PAuth is implemented:*

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- \( \text{APDAKeyHi}_\text{EL1} \).
- \( \text{APDAKeyLo}_\text{EL1} \).

<table>
<thead>
<tr>
<th>APDAKey</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of the System registers listed above are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or ( \text{SCR}_\text{EL3}.\text{FGTEn} == 1 ), MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.
Otherwise:

Reserved, RES0.

**AMAIR_EL1, bit [3]**

Trap MSR writes of AMAIR_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AMAIR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of AMAIR_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Bit [2]**

Reserved, RES0.

**AFSR1_EL1, bit [1]**

Trap MSR writes of AFSR1_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AFSR1_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of AFSR1_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**AFSR0_EL1, bit [0]**

Trap MSR writes of AFSR0_EL1 at EL1 using AArch64 to EL2.

<table>
<thead>
<tr>
<th>AFSR0_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSR writes of AFSR0_EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state and either EL3 is not implemented or SCR_EL3.FGTEn == 1, MSR writes of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

**Accessing the HFGWTR_EL2**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS &lt;Xt&gt;, HFGWTR_EL2</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
</tr>
<tr>
<td>0b11</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x1C0];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return HFGWTR_EL2;
elsif PSTATE.EL == EL3 then
  return HFGWTR_EL2;

MSR HFGWTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x1C0] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    HFGWTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  HFGWTR_EL2 = X[t];
The HPFAR_EL2 characteristics are:

**Purpose**

Holds the faulting IPA for some aborts on a stage 2 translation taken to EL2.

**Configuration**

AArch64 System register HPFAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HPFAR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HPFAR_EL2 is a 64-bit register.

**Field descriptions**

The HPFAR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS</td>
<td>RES0</td>
<td>FIPA[51:48]</td>
<td>FIPA[47:12]</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Execution at EL1 or EL0 makes HPFAR_EL2 become UNKNOWN.

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Faulting IPA address space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Faulting IPA is from the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>Faulting IPA is from the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

For data or instruction aborts taken to Non-secure EL2, this field is RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [62:44]**

Reserved, RES0.

**FIPA[51:48], bits [43:40]**
When ARMv8.2-LPA is implemented:


This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**FIPA[47:12], bits [39:4]**

Bits [47:12] of the faulting intermediate physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use for the stage 1 translation, the FIPA[51:48] bits form the upper part of the address value. For implementations or stage 1 translation granules with fewer than 52 physical address bits the FIPA[51:48] bits are RES0.

The HPFAR_EL2 is written for:

- Translation or Access faults in the second stage of translation.
- An abort in the second stage of translation performed during the translation table walk of a first stage translation, caused by a Translation fault, an Access flag fault, or a Permission fault.
- A stage 2 Address size fault.

**Note**

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lowest address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores a mis-aligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

For all other exceptions taken to EL2, this register is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**Bits [3:0]**

Reserved, RES0.

**Accessing the HPFAR_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, HPFAR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    return HPFAR_EL2;
elsif PSTATE.EL == EL3 then
    return HPFAR_EL2;

MSR HPFAR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    HPFAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HPFAR_EL2 = X[t];
HSTR_EL2, Hypervisor System Trap Register

The HSTR_EL2 characteristics are:

**Purpose**

Controls trapping to EL2 of EL1 or lower AArch32 accesses to the System register in the coproc == \[0b1111\] encoding space, by the CRn value used to access the register using MCR or MRC instruction. When the register is accessible using an MCRR or MRRC instruction, this is the CRm value used to access the register.

**Configuration**

AArch64 System register HSTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSTR[31:0].

If EL2 is not implemented, this register is `RES0` from EL3.

If no Exception level can use AArch32, then this register is `RES0`.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**

HSTR_EL2 is a 64-bit register.

**Field descriptions**

The HSTR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td><code>RES0</code></td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td><code>T15</code></td>
</tr>
<tr>
<td>42</td>
<td><code>T14</code></td>
</tr>
<tr>
<td>41</td>
<td><code>T13</code></td>
</tr>
<tr>
<td>40</td>
<td><code>T12</code></td>
</tr>
<tr>
<td>39</td>
<td><code>T11</code></td>
</tr>
<tr>
<td>38</td>
<td><code>T10</code></td>
</tr>
<tr>
<td>37</td>
<td><code>T9</code></td>
</tr>
<tr>
<td>36</td>
<td><code>T8</code></td>
</tr>
<tr>
<td>35</td>
<td><code>T7</code></td>
</tr>
<tr>
<td>34</td>
<td><code>T6</code></td>
</tr>
<tr>
<td>33</td>
<td><code>T5</code></td>
</tr>
<tr>
<td>32</td>
<td><code>T4</code></td>
</tr>
<tr>
<td>31</td>
<td><code>T3</code></td>
</tr>
<tr>
<td>30</td>
<td><code>T2</code></td>
</tr>
<tr>
<td>29</td>
<td><code>T1</code></td>
</tr>
<tr>
<td>28</td>
<td><code>T0</code></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, `RES0`.

T<\(n\)> , bit [\(n\) , for \(n = 0\) to \(15\)]

Fields T14 and T4 are `RES0`.

The remaining fields control whether EL0 and EL1 accesses, using MCR, MRC, MCRR, and MRRC instructions, to the System registers in the coproc == \[0b1111\] encoding space are trapped to EL2 as follows:

- MCR or MRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value \[0x03\], unless the access is `UNDEFINED`.
- MCRR or MRRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value \[0x04\], unless the access is `UNDEFINED`.

Page 760
This control has no effect on EL0 or EL1 accesses to System registers.

Any EL1 MCR or MRC access with coproc == 0b1111 and CRn == <n> is trapped to EL2. An EL0 MCR or MRC access with these values is trapped to EL2 only if the access is not UNDEFINED when the value of this field is 0.

Any EL1 MCRR or MRRC access with coproc == 0b1111 and CRm == <n> is trapped to EL2. An EL0 MCRR or MRRC access with these values is trapped to EL2 only if the access is not UNDEFINED when the value of this field is 0.

It is IMPLEMENTATION DEFINED whether a Non-secure EL0 access using AArch32 to these registers is trapped to EL2, or is UNDEFINED and generates an exception that is taken to Non-secure EL1. If the access is UNDEFINED, and generates an exception that is taken to Non-secure EL1 using AArch64, this is reported with EC syndrome value 0x00.

Note
Arm expects that trapping to EL2 of Non-secure EL0 accesses to these registers is unusual and used only when the hypervisor must virtualize EL0 operation. Arm recommends that, whenever possible, Non-secure EL0 accesses to these registers behave as they would if the implementation did not include EL2. This means that, if the architecture does not support the Non-secure EL0 access, then the register access instruction is treated as UNDEFINED and generates an exception that is taken to Non-secure EL1.

For example, when HSTR_EL2.T7 is 1, for instructions executed at EL1:

- An MCR or MRC instruction with coproc set to 0b1111 and <CRn> set to c7 is trapped to EL2.
- An MCRR or MRRC instruction with coproc set to 0b1111 and <CRm> set to c7 is trapped to EL2.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

This field resets to an architecturally UNKNOWN value.

**Accessing the HSTR_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, HSTR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x080];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        return HSTR_EL2;
    elsif PSTATE.EL == EL3 then
        return HSTR_EL2;
endif

MSR HSTR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x080] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        HSTR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        HSTR_EL2 = X[t];
IC IALLU, Instruction Cache Invalidate All to PoU

The IC IALLU characteristics are:

**Purpose**

Invalidate all instruction caches to Point of Unification.

**Configuration**

AArch64 System instruction IC IALLU performs the same function as AArch32 System instruction ICIALLU.

**Attributes**

IC IALLU is a 64-bit System instruction.

**Field descriptions**

IC IALLU ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the IC IALLU instruction**

Accesses to this instruction use the following encodings:

\[
\text{IC IALLU}\{, \langle Xt \rangle\}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b000</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.ICIALLU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        IC_IALLUIS();
    else
        IC_IALLU();
    endif
elsif PSTATE.EL == EL2 then
    IC_IALLU();
else
    IC_IALLU();
elesif PSTATE.EL == EL3 then
    IC_IALLU();
elesif PSTATE.EL == EL4 then
    IC_IALLU();
else
    IC_IALLU();
elesif PSTATE.EL == EL5 then
    IC_IALLU();
elesif PSTATE.EL == EL6 then
    IC_IALLU();
elesif PSTATE.EL == EL7 then
    IC_IALLU();
elesif PSTATE.EL == EL8 then
    IC_IALLU();
elesif PSTATE.EL == EL9 then
    IC_IALLU();
elesif PSTATE.EL == EL10 then
    IC_IALLU();
elesif PSTATE.EL == EL11 then
    IC_IALLU();
elesif PSTATE.EL == EL12 then
    IC_IALLU();
elesif PSTATE.EL == EL13 then
    IC_IALLU();
elesif PSTATE.EL == EL14 then
    IC_IALLU();
elesif PSTATE.EL == EL15 then
    IC_IALLU();
else
    IC_IALLU();
IC IALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable

The IC IALLUIS characteristics are:

**Purpose**

Invalidate all instruction caches in Inner Shareable domain to Point of Unification.

**Configuration**

AArch64 System instruction IC IALLUIS performs the same function as AArch32 System instruction ICIALLUIS.

**Attributes**

IC IALLUIS is a 64-bit System instruction.

**Field descriptions**

IC IALLUIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the IC IALLUIS instruction**

Accesses to this instruction use the following encodings:

```
IC IALLUIS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b11111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TICAB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.ICIALLUIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        IC_IALLUIS();
    endif
elsif PSTATE.EL == EL2 then
    IC_IALLUIS();
elsif PSTATE.EL == EL3 then
    IC_IALLUIS();
IC IVAU, Instruction Cache line Invalidate by VA to PoU

The IC IVAU characteristics are:

**Purpose**

Invalidate instruction cache by address to Point of Unification.

**Configuration**

AArch64 System instruction IC IVAU performs the same function as AArch32 System instruction ICIMVAU.

**Attributes**

IC IVAU is a 64-bit System instruction.

**Field descriptions**

The IC IVAU input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the IC IVAU instruction**

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The instruction cache maintenance instruction (IC)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it is IMPLEMENTATION DEFINED whether it generates a Permission Fault, see 'Permission fault' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

IC IVAU{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL1.UCI == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    endif
elsif !EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif !EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPU == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif !EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TOCU == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif !EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ICIVAU == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif !EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
else
    IC_IVAU(X[t]);
endif
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ICIVAU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SC...
The ICC_AP0R<n>_EL1 characteristics are:

**Purpose**

Provides information about Group 0 active priorities.

**Configuration**

AArch64 System register ICC_AP0R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_AP0R<n>[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_AP0R<n>_EL1 is a 64-bit register.

**Field descriptions**

The ICC_AP0R<n>_EL1 bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| RES0   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| RES0   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |

**Bits [63:32]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

**Accessing the ICC_AP0R<n>_EL1**

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP0R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICC_AP0R2_EL1 and ICC_AP0R3_EL1 are only implemented in implementations that support 7 or more bits of priority. Unimplemented registers are UNDEFINED.

**Note**
The number of bits of preemption is indicated by \texttt{ICH\_VTR\_EL2\_PREbits}.

Writing to the active priority registers in any order other than the following order will result in \textsc{unpredictable} behavior:

- \texttt{ICC\_AP0R<n>_EL1}.
- Secure \texttt{ICC\_AP1R<n>_EL1}.
- Non-secure \texttt{ICC\_AP1R<n>_EL1}.

Accesses to this register use the following encodings:

\textbf{MRS <Xt>, ICC\_AP0R<n>_EL1}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
</tr>
</tbody>
</table>

\begin{verbatim}
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICC_AP0R_EL1[UInt(op2<1:0>)];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_AP0R_EL1[UInt(op2<1:0>)];
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_AP0R_EL1[UInt(op2<1:0>)];
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_AP0R_EL1[UInt(op2<1:0>)];
    end
\end{verbatim}

\textbf{MSR ICC\_AP0R<n>_EL1, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_AP0R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
    endif
else
    ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
    endif
ICC_AP1R<n>_EL1, Interrupt Controller Active Priorities Group 1 Registers, n = 0 - 3

The ICC_AP1R<n>_EL1 characteristics are:

**Purpose**

Provides information about Group 1 active priorities.

**Configuration**

AArch64 System register ICC_AP1R<n>_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register ICC_AP1R<n>[31:0] (S).

AArch64 System register ICC_AP1R<n>_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register ICC_AP1R<n>[31:0] (NS).

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_AP1R<n>_EL1 is a 64-bit register.

**Field descriptions**

The ICC_AP1R<n>_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:32]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

**Accessing the ICC_AP1R<n>_EL1**

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP1R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICC_AP1R2_EL1 and ICC_AP1R3_EL1 are only implemented in implementations that support 7 or more bits of priority. Unimplemented registers are UNDEFINED.
Note

The number of bits of preemption is indicated by `ICH_VTR_EL2.PREbits`.

Writing to the active priority registers in any order other than the following order will result in **UNPREDICTABLE** behavior:

- `ICC_AP0R<n>_EL1`
- Secure `ICC_AP1R<n>_EL1`
- Non-secure `ICC_AP1R<n>_EL1`

Accesses to this register use the following encodings:

**MRS <Xt>, ICC_AP1R<n>_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_AP1R_EL1[UInt(op2<1:0>)];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
    else
      return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
  else
    return ICC_AP1R_EL1[UInt(op2<1:0>)];
  end if
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
    else
      return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
  else
    return ICC_AP1R_EL1[UInt(op2<1:0>)];
  end if
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    if SCR_EL3.NS == '0' then
      return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
    else
      return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
  end if
else
  return ICC_AP1R_EL1[UInt(op2<1:0>)];
end if

**MSR ICC_AP1R<n>_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x10);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x10);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x10);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x10);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x10);
    elseif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x10);
    else
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
The ICC_ASGI1R_EL1 characteristics are:

**Purpose**

Generates Group 1 SGIs for the Security state that is not the current Security state.

**Configuration**

AArch64 System register ICC_ASGI1R_EL1 performs the same function as AArch32 System register ICC_ASGI1R.

Under certain conditions a write to ICC_ASGI1R_EL1 can generate Group 0 interrupts, see "Forwarding an SGI to a target PE".

**Attributes**

ICC_ASGI1R_EL1 is a 64-bit register.

**Field descriptions**

The ICC_ASGI1R_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>Aff3</td>
</tr>
<tr>
<td>61</td>
<td>Aff2</td>
</tr>
<tr>
<td>60</td>
<td>INTID</td>
</tr>
<tr>
<td>59</td>
<td>Aff1</td>
</tr>
<tr>
<td>58</td>
<td>TargetList</td>
</tr>
<tr>
<td>57</td>
<td>RES0</td>
</tr>
<tr>
<td>56</td>
<td>RM</td>
</tr>
<tr>
<td>55</td>
<td>Aff2</td>
</tr>
<tr>
<td>54</td>
<td>RS</td>
</tr>
<tr>
<td>53</td>
<td>RES0</td>
</tr>
<tr>
<td>52</td>
<td>Aff2</td>
</tr>
<tr>
<td>51</td>
<td>Aff1</td>
</tr>
<tr>
<td>50</td>
<td>TargetList</td>
</tr>
<tr>
<td>49</td>
<td>RES0</td>
</tr>
<tr>
<td>48</td>
<td>Aff2</td>
</tr>
<tr>
<td>47</td>
<td>Aff1</td>
</tr>
<tr>
<td>46</td>
<td>TargetList</td>
</tr>
<tr>
<td>45</td>
<td>RES0</td>
</tr>
<tr>
<td>44</td>
<td>Aff2</td>
</tr>
<tr>
<td>43</td>
<td>Aff1</td>
</tr>
<tr>
<td>42</td>
<td>TargetList</td>
</tr>
<tr>
<td>41</td>
<td>RES0</td>
</tr>
<tr>
<td>40</td>
<td>Aff2</td>
</tr>
<tr>
<td>39</td>
<td>Aff1</td>
</tr>
<tr>
<td>38</td>
<td>TargetList</td>
</tr>
<tr>
<td>37</td>
<td>RES0</td>
</tr>
<tr>
<td>36</td>
<td>Aff2</td>
</tr>
<tr>
<td>35</td>
<td>Aff1</td>
</tr>
<tr>
<td>34</td>
<td>TargetList</td>
</tr>
<tr>
<td>33</td>
<td>RES0</td>
</tr>
<tr>
<td>32</td>
<td>Aff2</td>
</tr>
</tbody>
</table>

**Bits [63:56]**

Reserved, RES0.

**Aff3, bits [55:48]**

The affinity 3 value of the affinity path of the luster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**RS, bits [47:44]**

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value (RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.
Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

<table>
<thead>
<tr>
<th>IRM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.&lt;target list&gt;.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to all PEs in the system, excluding &quot;self&quot;.</td>
</tr>
</tbody>
</table>

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

This restricts a system to sending targeted SGIs to PEs with an affinity 0 number that is less than 16. If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing the ICC_ASGI1R_EL1

This register allows software executing in a Secure state to generate Non-secure Group 1 SGIs. It will also allow software executing in a Non-secure state to generate Secure Group 1 SGIs, if permitted by the settings of GICR_NSACR in the Redistributor corresponding to the target PE.
When `GICD_CTLR.DS==0`, Non-secure writes do not generate an interrupt for a target PE if not permitted by the `GICR_NSACR` register associated with the target PE. For more information see Use of control registers for SGI forwarding.

**Note**

Accesses at EL3 are treated as Secure regardless of the value of `SCR_EL3.NS`.

Accesses to this register use the following encodings:

MSR `ICC_ASGI1R_EL1`, `<Xt>`

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_ASGI1R_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_ASGI1R_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_ASGI1R_EL1 = X[t];
    endif
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0

The ICC_BPR0_EL1 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

**Configuration**

AArch64 System register ICC_BPR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_BPR0[31:0].

Virtual accesses to this register update ICH_VMCR_EL2.VBPR0.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_BPR0_EL1 is a 64-bit register.

**Field descriptions**

The ICC_BPR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Group Priority Field</th>
<th>Subpriority Field</th>
<th>Field with Binary Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>BinaryPoint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>0</td>
<td>[7:1]</td>
<td>0</td>
<td>ggggggg.s</td>
</tr>
<tr>
<td>60</td>
<td>1</td>
<td>[7:2]</td>
<td>1</td>
<td>gggggss</td>
</tr>
<tr>
<td>59</td>
<td>2</td>
<td>[7:3]</td>
<td>2</td>
<td>ggggg.gss</td>
</tr>
<tr>
<td>58</td>
<td>3</td>
<td>[7:4]</td>
<td>3</td>
<td>ggg.gssss</td>
</tr>
<tr>
<td>57</td>
<td>4</td>
<td>[7:5]</td>
<td>4</td>
<td>gssssss</td>
</tr>
<tr>
<td>56</td>
<td>5</td>
<td>[7:6]</td>
<td>5</td>
<td>gg.sssss</td>
</tr>
<tr>
<td>55</td>
<td>6</td>
<td>[7]</td>
<td>6</td>
<td>g.sssssss</td>
</tr>
<tr>
<td>54</td>
<td>7</td>
<td>No preemption</td>
<td>7</td>
<td>.ssssssss</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
Accessing the ICC_BPR0_EL1

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is implementation defined, and reported by ICC_CTLR_EL1_PRIbits and ICC_CTLR_EL3_PRIbits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is unknown.

Accesses to this register use the following encodings:

MRS <Xt>, ICC_BPR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_BPR0_EL1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_BPR0_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_BPR0_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_BPR0_EL1;

MSR ICC_BPR0_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_BPR0_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
    end if;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
    end if;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
    end if;
ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1

The ICC_BPR1_EL1 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

**Configuration**

AArch64 System register ICC_BPR1_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register ICC_BPR1[31:0] (S).

AArch64 System register ICC_BPR1_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register ICC_BPR1[31:0] (NS).

Virtual accesses to this register update ICH_VMCR_EL2.VBPR1.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_BPR1_EL1 is a 64-bit register.

**Field descriptions**

The ICC_BPR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | BinaryPoint |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:3]**

Reserved, RES0.

**BinaryPoint, bits [2:0]**

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. For more information about priorities, see Priority grouping.

The minimum value of the Non-secure copy of this register is the minimum value of ICC_BPR0_EL1 + 1. The minimum value of the Secure copy of this register is the minimum value of ICC_BPR0_EL1.

If EL3 is implemented and ICC_CTLR_EL3.CBPR_EL1S is 1:

- When SCR_EL3.EEL2 is 1 and HCR_EL2.IMO is 1, Secure accesses to this register at EL1 access the state of ICV_BPR1_EL1.
- Otherwise, Secure accesses to this register at EL1 access the state of ICC_BPR0_EL1.

If EL3 is implemented and ICC_CTLR_EL3.CBPR_EL1NS is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR_EL2.IMO and SCR_EL3.IRQ:
If EL3 is not implemented and ICC_CTLR_EL1.CBPR is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR_EL2.IMO:

<table>
<thead>
<tr>
<th>HCR_EL2.IMO</th>
<th>SCR_EL3.IRQ</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b0</td>
<td>Non-secure EL1 and EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored.</td>
</tr>
<tr>
<td>0b0</td>
<td>0b1</td>
<td>Non-secure EL1 and EL2 accesses trap to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL2 writes are ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 accesses trap to EL3.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

### Accessing the ICC_BPR1_EL1

On a reset, the binary point field is UNKNOWN.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

Accesses to this register use the following encodings:

MRS <Xt>, ICC_BPR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !EL3UsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL2.IRQ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) then
        if SCR_EL2.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
        end if
    end if
else
    return ICC_BPR1_EL1;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !EL3UsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
        end if
    else
    return ICC_BPR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
        end if
    end if
end if

MSR ICC_BPR1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICC_BPR1_EL1 = X[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      ICC_BPR1_EL1_S = X[t];
    else
      ICC_BPR1_EL1_NS = X[t];
  else
    ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      ICC_BPR1_EL1_S = X[t];
    else
      ICC_BPR1_EL1_NS = X[t];
  else
    ICC_BPR1_EL1 = X[t];
else
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    if SCR_EL3.NS == '0' then
      ICC_BPR1_EL1_S = X[t];
    else
      ICC_BPR1_EL1_NS = X[t];

ICC_CTLR_EL1, Interrupt Controller Control Register (EL1)

The ICC_CTLR_EL1 characteristics are:

**Purpose**

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

**Configuration**

AArch64 System register ICC_CTLR_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register ICC_CTLR[31:0] (S).

AArch64 System register ICC_CTLR_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register ICC_CTLR[31:0] (NS).

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_CTLR_EL1 is a 64-bit register.

**Field descriptions**

The ICC_CTLR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:20]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
</table>

**ExtRange, bit [19]**

Extended INTID range (read-only).

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>CPU interface does not support INTIDs in the range 1024..8191.</td>
</tr>
<tr>
<td>0b1</td>
<td>CPU interface supports INTIDs in the range 1024..8191</td>
</tr>
</tbody>
</table>

Note

Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.

If EL3 is implemented, ICC_CTLR_EL1.ExtRange is an alias of ICC_CTLR_EL3.ExtRange.
RSS, bit [18]

Range Selector Support. Possible values are:

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 15 are supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 255 are supported.</td>
</tr>
</tbody>
</table>

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
</tbody>
</table>

If EL3 is implemented, this bit is an alias of ICC_CTLR_EL3.A3V.

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports local generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic does not support local generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports local generation of SEIs.</td>
</tr>
</tbody>
</table>

If EL3 is implemented, this bit is an alias of ICC_CTLR_EL3.SEIS.

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b0001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If EL3 is implemented, this field is an alias of ICC_CTLR_EL3.IDbits.

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the Security state of the access or the value of GICD_CTRLR.DS.
For physical accesses, this field determines the minimum value of \texttt{ICC\_BPR0\_EL1}.

If EL3 is implemented, physical accesses return the value from \texttt{ICC\_CTRLR\_EL3} PRI bits.

If EL3 is not implemented, physical accesses return the value from this field.

**Bit [7]**

Reserved, RES0.

**PMHE, bit [6]**

Priority Mask Hint Enable. Controls whether the priority mask register is used as a hint for interrupt distribution:

<table>
<thead>
<tr>
<th>PMHE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables use of ICC_PMR_EL1 as a hint for interrupt distribution.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables use of ICC_PMR_EL1 as a hint for interrupt distribution.</td>
</tr>
</tbody>
</table>

If EL3 is implemented, this bit is an alias of ICC\_CTRLR\_EL3 PMHE. Whether this bit can be written as part of an access to this register depends on the value of GICD\_CTRLR\_DS:

- If GICD\_CTRLR\_DS == 0, this bit is read-only.
- If GICD\_CTRLR\_DS == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

**Bits [5:2]**

Reserved, RES0.

**EOImode, bit [1]**

EOI mode for the current Security state. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

The Secure ICC\_CTRLR\_EL1.EOIMode is an alias of ICC\_CTRLR\_EL3.EOIMode\_EL1S.

The Non-secure ICC\_CTRLR\_EL1.EOIMode is an alias of ICC\_CTRLR\_EL3.EOIMode\_EL1NS

**CBPR, bit [0]**

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupts:

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_BPR0_EL1 determines the preemption group for both Group 0 and Group 1 interrupts.</td>
</tr>
</tbody>
</table>
If EL3 is implemented:

- This bit is an alias of ICC_CTLR_EL3.CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If GICD_CTLR.DS == 0, this bit is read-only.
- If GICD_CTLR.DS == 1, this bit is read/write.

If EL3 is not implemented, this bit is read/write.

This field resets to an architecturally UNKNOWN value.

**Accessing the ICC_CTLR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ICC_CTLR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICC_CTLR_EL1;
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICC_CTLR_EL1;
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elseif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      return ICC_CTLR_EL1_S;
    else
      return ICC_CTLR_EL1_NS;
  else
    return ICC_CTLR_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elseif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      return ICC_CTLR_EL1_S;
    else
      return ICC_CTLR_EL1_NS;
  else
    return ICC_CTLR_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elseif if SCR_EL3.NS == '0' then
    return ICC_CTLR_EL1_S;
  else
    return ICC_CTLR_EL1_NS;
else
  MSR ICC_CTLR_EL1, <Xt>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICC_CTLR_EL1 = X[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICC_CTLR_EL1 = X[t];
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
        end
    else
        ICC_CTLR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
        end
    else
        ICC_CTLR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif SCR_EL3.NS == '0' then
        ICC_CTLR_EL1_S = X[t];
    else
        ICC_CTLR_EL1_NS = X[t];
    end
ICC_CTLR_EL3, Interrupt Controller Control Register (EL3)

The ICC_CTLR_EL3 characteristics are:

**Purpose**

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

**Configuration**

AArch64 System register ICC_CTLR_EL3 bits [31:0] can be mapped to AArch32 System register ICC_MCTRL[31:0], but this is not architecturally mandated.

This register is present only when HaveEL(EL3). Otherwise, direct accesses to ICC_CTLR_EL3 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_CTLR_EL3 is a 64-bit register.

**Field descriptions**

The ICC_CTLR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Range</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:20</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>ExtRange</td>
<td>Extended INTID range (read-only).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>CPU interface does not support INTIDs in the range 1024..8191.</td>
</tr>
<tr>
<td></td>
<td>• Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.</td>
</tr>
<tr>
<td></td>
<td>Note</td>
</tr>
<tr>
<td></td>
<td>Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.</td>
</tr>
<tr>
<td>0b1</td>
<td>CPU interface supports INTIDs in the range 1024..8191.</td>
</tr>
<tr>
<td></td>
<td>• All INTIDs in the range 1024..8191 are treated as requiring deactivation.</td>
</tr>
</tbody>
</table>
RSS, bit [18]

Range Selector Support. Possible values are:

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 15 are supported.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 255 are supported.</td>
<td></td>
</tr>
</tbody>
</table>

This bit is read-only.

nDS, bit [17]

Disable Security not supported. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>nDS</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic supports disabling of security.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic does not support disabling of security, and requires that security is not disabled.</td>
<td></td>
</tr>
</tbody>
</table>

Bit [16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic does not support non-zero values of the Aff3 field in SGI generation System registers.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports non-zero values of the Aff3 field in SGI generation System registers.</td>
<td></td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTLR_EL1.AV3 is an alias of ICC_CTLR_EL3.A3V

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic does not support generation of SEIs.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports generation of SEIs.</td>
<td></td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTLR_EL1.SEIS is an alias of ICC_CTLR_EL3.SEIS

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
<td></td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved.

If EL3 is present, ICC_CTLR_EL1.IDbits is an alias of ICC_CTLR_EL3.IDbits

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).
An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

**Note**

This field always returns the number of priority bits implemented, regardless of the value of SCR_EL3.NS or the value of GICD_CTLR.DS.

The division between group priority and subpriority is defined in the binary point registers ICC_BPR0_EL1 and ICC_BPR1_EL1.

This field determines the minimum value of ICC_BPR0_EL1.

**Bit [7]**

Reserved, RES0.

**PMHE, bit [6]**

Priority Mask Hint Enable.

<table>
<thead>
<tr>
<th>PMHE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables use of the priority mask register as a hint for interrupt distribution.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables use of the priority mask register as a hint for interrupt distribution.</td>
</tr>
</tbody>
</table>

Software must write ICC_PMR_EL1 to 0xFF before clearing this field to 0.

- An implementation might choose to make this field RAO/WI if priority-based routing is always used
- An implementation might choose to make this field RAZ/WI if priority-based routing is never used

If EL3 is present, ICC_CTLR_EL1.PMHE is an alias of ICC_CTLR_EL3.PMHE.

This field resets to 0.

**RM, bit [5]**

Routing Modifier. For legacy operation of EL1 software with GICC_CTLR.FIQen set to 1, this bit indicates whether interrupts can be acknowledged or observed as the Highest Priority Pending Interrupt, or whether a special INTID value is returned.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>RM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure Group 0 and Non-secure Group 1 interrupts can be acknowledged and observed as the highest priority interrupt at the Secure Exception level where the interrupt is taken.</td>
</tr>
</tbody>
</table>
| 0b1 | When accessed at EL3 in AArch64 state:
  - Secure Group 0 interrupts return a special INTID value of 1020. This affects accesses to ICC_IAR0_EL1 and ICC_HPPIR0_EL1.
  - Non-secure Group 1 interrupts return a special INTID value of 1021. This affects accesses to ICC_IAR1_EL1 and ICC_HPPIR1_EL1. |

**Note**

The Routing Modifier bit is supported in AArch64 only. In systems without EL3 the behavior is as if the value is 0. Software must ensure this bit is 0 when the Secure copy of ICC_SRE_EL1.SRE is 1, otherwise system behavior is UNPREDICTABLE. In systems without EL3 or where the Secure copy of ICC_SRE_EL1.SRE is RAO/WI, this bit is RES0.

This field resets to an architecturally UNKNOWN value.
EOImode_EL1NS, bit [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode_EL1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTRLR_EL1(NS).EOImode is an alias of ICC_CTRLR_EL3.EOImode_EL1NS.

This field resets to an architecturally UNKNOWN value.

EOImode_EL1S, bit [3]

EOI mode for interrupts handled at Secure EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode_EL1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTRLR_EL1(S).EOImode is an alias of ICC_CTRLR_EL3.EOImode_EL1S.

This field resets to an architecturally UNKNOWN value.

EOImode_EL3, bit [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

CBPR_EL1NS, bit [1]

Common Binary Point Register, EL1 Non-secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2:

<table>
<thead>
<tr>
<th>CBPR_EL1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Non-secure Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts and Non-secure Group 1 interrupts. Non-secure accesses to GICC_BPR and ICC_BPR1_EL1 access the state of ICC_BPR0_EL1.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTRLR_EL1(NS).CBPR is an alias of ICC_CTRLR_EL3.CBPR_EL1NS.
This field resets to an architecturally **UNKNOWN** value.

**CBPR_EL1S, bit [0]**

Common Binary Point Register, EL1 Secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupts at EL1:

<table>
<thead>
<tr>
<th>CBPR_EL1S</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0       | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only.  
           | ICC_BPR1_EL1 determines the preemption group for Secure Group 1 interrupts. |
| 0b1       | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts and Secure Group 1 interrupts.  
           | Secure EL1 accesses to ICC_BPR1_EL1 access the state of ICC_BPR0_EL1. |

If EL3 is present, ICC_CTLR_EL1(S).CBPR is an alias of ICC_CTLR_EL3.CBPR_EL1S.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ICC_CTLR_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, ICC_CTLR_EL3

```assembly
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
```

```assembly
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_CTLR_EL3;

MSR ICC_CTLR_EL3, <Xt>

```assembly
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
```

```assembly
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_CTLR_EL3 = X[t];
```
ICC_DIR_EL1, Interrupt Controller Deactivate Interrupt Register

The ICC_DIR_EL1 characteristics are:

**Purpose**

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

**Configuration**

AArch64 System register ICC_DIR_EL1 performs the same function as AArch32 System register ICC_DIR.

**Attributes**

ICC_DIR_EL1 is a 64-bit register.

**Field descriptions**

The ICC_DIR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23-0</td>
<td>INTID, bits 23-0</td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_DIR_EL1**

There are two cases when writing to ICC_DIR_EL1 that were UNPREDICTABLE for a corresponding GICv2 write to GICC_DIR:

- When EOImode == 0. GICv3 implementations must ignore such writes. In systems supporting system error generation, an implementation might generate an SEI.
- When EOImode == 1 but no EOI has been issued. The interrupt will be de-activated by the Distributor, however the active priority in the CPU interface for the interrupt will remain set (because no EOI was issued).

Accesses to this register use the following encodings:

**MSR ICC_DIR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TDIR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_DIR_EL1 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_DIR_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];
The ICC_EOIR0_EL1 characteristics are:

**Purpose**

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 0 interrupt.

**Configuration**

AArch64 System register ICC_EOIR0_EL1 performs the same function as AArch32 System register ICC_EOIR0.

**Attributes**

ICC_EOIR0_EL1 is a 64-bit register.

**Field descriptions**

The ICC_EOIR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>INTID from the corresponding ICC_IAR0_EL1 access.</td>
</tr>
<tr>
<td>48</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>47</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>46</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>45</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>44</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>43</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>42</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>41</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>40</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>39</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>38</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>37</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>36</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>35</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>34</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>33</td>
<td>INTID [23:0].</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID from the corresponding ICC_IAR0_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1_IDbits and ICC_CTLR_EL3_IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR_EL1 to deactivate the interrupt.

The EOImode bit for the current Exception level and Security state is determined as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR_EL1 EOImode.
- If EL3 is implemented and the software is executing at EL3, the appropriate bit is ICC_CTLR_EL3 EOImode_EL3.
- If EL3 is implemented and the software is not executing at EL3, the bit depends on the current Security state:
  - If the software is executing in Secure state, the bit is ICC_CTLR_EL3 EOImode_EL1S.
  - If the software is executing in Non-secure state, the bit is ICC_CTLR_EL3 EOImode_EL1NS.
Accessing the \textbf{ICC\_EOIR0\_EL1}

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from \textbf{ICC\_IAR0\_EL1}, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. See Special INTIDs, for more information.

Accesses to this register use the following encodings:

\begin{verbatim}
MSR ICC\_EOIR0\_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC\_SRE\_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH\_HCR\_EL2.TALL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR\_EL2.FMO == '1' then
    ICV\_EOIR0\_EL1 = X\{t\};
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR\_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC\_EOIR0\_EL1 = X\{t\};
elsif PSTATE.EL == EL2 then
  if ICC\_SRE\_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR\_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC\_EOIR0\_EL1 = X\{t\};
elsif PSTATE.EL == EL3 then
  if ICC\_SRE\_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC\_EOIR0\_EL1 = X\{t\};
\end{verbatim}

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ICC_EOIR1_EL1, Interrupt Controller End Of Interrupt Register 1

The ICC_EOIR1_EL1 characteristics are:

**Purpose**

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 1 interrupt.

**Configuration**

AArch64 System register ICC_EOIR1_EL1 performs the same function as AArch32 System register ICC_EOIR1.

**Attributes**

ICC_EOIR1_EL1 is a 64-bit register.

**Field descriptions**

The ICC_EOIR1_EL1 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     | RES0| RES0| INTID|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID from the corresponding ICC_IAR1_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR_EL1 to deactivate the interrupt.

The EOImode bit for the current Exception level and Security state is determined as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR_EL1.EOImode.
- If EL3 is implemented and the software is executing at EL3, the appropriate bit is ICC_CTLR_EL3.EOImode_EL3.
- If EL3 is implemented and the software is not executing at EL3, the bit depends on the current Security state:
  - If the software is executing in Secure state, the bit is ICC_CTLR_EL3.EOImode_EL1S.
  - If the software is executing in Non-secure state, the bit is ICC_CTLR_EL3.EOImode_EL1NS.
### Accessing the ICC_EOIR1_EL1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR1_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. See Special INTIDs, for more information.

Accesses to this register use the following encodings:

MSR ICC_EOIR1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_EOIR1_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
    endif
endif
```

27/09/2019 18:22; 6134483bd1d414dc8c1a99e984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The ICC_HPPIR0_EL1 characteristics are:

**Purpose**

Indicates the highest priority pending Group 0 interrupt on the CPU interface.

**Configuration**

AArch64 System register ICC_HPPIR0_EL1 performs the same function as AArch32 System register ICC_HPPIR0.

**Attributes**

ICC_HPPIR0_EL1 is a 64-bit register.

**Field descriptions**

The ICC_HPPIR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>RES0</td>
<td>INTID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_HPPIR0_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ICC_HPPIR0_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsf PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICC_HPPIR0_EL1;
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ICC_HPPIR0_EL1;
elsf PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ICC_HPPIR0_EL1;
elsf PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ICC_HPPIR0_EL1;
**ICC_HPPIR1_EL1, Interrupt Controller Highest Priority Pending Interrupt Register 1**

The ICC_HPPIR1_EL1 characteristics are:

**Purpose**

Indicates the highest priority pending Group 1 interrupt on the CPU interface.

**Configuration**

AArch64 System register ICC_HPPIR1_EL1 performs the same function as AArch32 System register ICC_HPPIR1.

**Attributes**

ICC_HPPIR1_EL1 is a 64-bit register.

**Field descriptions**

The ICC_HPPIR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23-0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_HPPIR1_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ICC_HPPIR1_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICC_HPPIR1_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL2 then
        if ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;
The ICC_IAR0_EL1 characteristics are:

**Purpose**

The PE reads this register to obtain the INTID of the signaled Group 0 interrupt. This read acts as an acknowledge for the interrupt.

**Configuration**

AArch64 System register ICC_IAR0_EL1 performs the same function as AArch32 System register ICC_IAR0.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.\{I,F\} == \{0,0\}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

**Attributes**

ICC_IAR0_EL1 is a 64-bit register.

**Field descriptions**

The ICC_IAR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Index</th>
<th>Bit Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-24</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>23-0</td>
<td>INTID</td>
<td>INTID of the signaled interrupt. If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. See Special INTIDs, for more information.</td>
</tr>
</tbody>
</table>

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_IAR0_EL1**

Accesses to this register use the following encodings:
MRS <Xt>, ICC_IAR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if ICC_SRE_EL1.SRE == '0' then
      AArch64.SystemAccessTrap(EL1, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
      return ICC_IAR0_EL1;
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   else
      return ICC_IAR0_EL1;
   elsif PSTATE.EL == EL2 then
      if ICC_SRE_EL2.SRE == '0' then
         AArch64.SystemAccessTrap(EL2, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         return ICC_IAR0_EL1;
      elsif PSTATE.EL == EL3 then
         if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
         else
            return ICC_IAR0_EL1;
         end if;
      end if;
   end if;
The ICC_IAR1_EL1 characteristics are:

**Purpose**

The PE reads this register to obtain the INTID of the signaled Group 1 interrupt. This read acts as an acknowledge for the interrupt.

**Configuration**

AArch64 System register ICC_IAR1_EL1 performs the same function as AArch32 System register ICC_IAR1.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

**Attributes**

ICC_IAR1_EL1 is a 64-bit register.

**Field descriptions**

The ICC_IAR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | INTID | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 |

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR_EL1.IDbits and ICC_CTLR_EL3.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_IAR1_EL1**

Accesses to this register use the following encodings:
MRS <Xt>, ICC_IAR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICC_IAR1_EL1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR1_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR1_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR1_EL1;
The ICC_IGRPEN0_EL1 characteristics are:

**Purpose**

Controls whether Group 0 interrupts are enabled or not.

**Configuration**

AArch64 System register ICC_IGRPEN0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_IGRPEN0[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_IGRPEN0_EL1 is a 64-bit register.

**Field descriptions**

The ICC_IGRPEN0_EL1 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |     |     |     |   |   |
| RES0 | RES0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |     |     |     |   |   |
| Bits [63:1] |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

Reserved, RES0.

**Enable, bit [0]**

Enables Group 0 interrupts.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

Virtual accesses to this register update ICH_VMCR_EL2.VENG0.

If the highest priority pending interrupt for that PE is a Group 0 interrupt using 1 of N model, then the interrupt will be targeted to another PE as a result of the Enable bit changing from 1 to 0.

This field resets to 0.

**Accessing the ICC_IGRPEN0_EL1**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICC_IGRPEN0_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN0_EL1;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN0_EL1;

MSR ICC_IGRPEN0_EL1, <Xt>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICC_IGRPEN0_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t];
    endif
The ICC_IGRPEN1_EL1 characteristics are:

**Purpose**

Controls whether Group 1 interrupts are enabled for the current Security state.

**Configuration**

AArch64 System register ICC_IGRPEN1_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register ICC_IGRPEN1[31:0] (S).

AArch64 System register ICC_IGRPEN1_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register ICC_IGRPEN1[31:0] (NS).

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_IGRPEN1_EL1 is a 64-bit register.

**Field descriptions**

The ICC_IGRPEN1_EL1 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| RES0|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [63:1]**

Reserved, RES0.

**Enable, bit [0]**

Enables Group 1 interrupts for the current Security state.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 interrupts are disabled for the current Security state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 interrupts are enabled for the current Security state.</td>
</tr>
</tbody>
</table>

Virtual accesses to this register update ICH_VMCR_EL2.VENG1.

If EL3 is present:

- The Secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1S bit.
- The Non-secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

This field resets to 0.
**Accessing the ICC_IGRPEN1_EL1**

Accesses to this register use the following encodings:

MRS &lt;Xt&gt;, ICC_IGRPEN1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if ICC_SRE_EL1.SRE == '0' then
      AArch64.SystemAccessTrap(EL1, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGRTR_EL2.ICC_IGRPENn_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
      return ICV_IGRPEN1_EL1;
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   elsif HaveEL(EL3) then
      if SCR_EL3.NS == '0' then
         return ICC_IGRPEN1_EL1_S;
      else
         return ICC_IGRPEN1_EL1_NS;
   else
      return ICC_IGRPEN1_EL1;
   elsif PSTATE.EL == EL2 then
      if ICC_SRE_EL2.SRE == '0' then
         AArch64.SystemAccessTrap(EL2, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      elsif HaveEL(EL3) then
         if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
         else
            return ICC_IGRPEN1_EL1_NS;
      else
         return ICC_IGRPEN1_EL1;
   elsif PSTATE.EL == EL3 then
      if ICC_SRE_EL3.SRE == '0' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
         else
            return ICC_IGRPEN1_EL1_NS;
   else
      MSR ICC_IGRPEN1_EL1, &lt;Xt&gt;

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

Page 812
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICC_IGRPEN1_EL1 = X[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      ICC_IGRPEN1_EL1_S = X[t];
    else
      ICC_IGRPEN1_EL1_NS = X[t];
    else
      ICC_IGRPEN1_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
      if SCR_EL3.NS == '0' then
        ICC_IGRPEN1_EL1_S = X[t];
      else
        ICC_IGRPEN1_EL1_NS = X[t];
      else
        ICC_IGRPEN1_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      if SCR_EL3.NS == '0' then
        ICC_IGRPEN1_EL1_S = X[t];
      else
        ICC_IGRPEN1_EL1_NS = X[t];
    else
      ICC_IGRPEN1_EL1 = X[t];
ICC_IGRPEN1_EL3, Interrupt Controller Interrupt Group 1 Enable register (EL3)

The ICC_IGRPEN1_EL3 characteristics are:

**Purpose**

Controls whether Group 1 interrupts are enabled or not.

**Configuration**

AArch64 System register ICC_IGRPEN1_EL3 bits [31:0] can be mapped to AArch32 System register ICC_MGRPEN1[31:0], but this is not architecturally mandated.

This register is present only when HaveEL(EL3). Otherwise, direct accesses to ICC_IGRPEN1_EL3 are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICC_IGRPEN1_EL3 is a 64-bit register.

**Field descriptions**

The ICC_IGRPEN1_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>EnableGrp1S</td>
<td>Enables Group 1 interrupts for the Secure state.</td>
</tr>
<tr>
<td>61</td>
<td>EnableGrp1NS</td>
<td>Enables Group 1 interrupts for the Non-secure state.</td>
</tr>
</tbody>
</table>

**Bits [63:2]**

Reserved, RES0.

**EnableGrp1S, bit [1]**

Enables Group 1 interrupts for the Secure state.

<table>
<thead>
<tr>
<th>EnableGrp1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Secure Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>1</td>
<td>Secure Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

The Secure ICC_IGRPEN1_EL1.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1S bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

This field resets to 0.

**EnableGrp1NS, bit [0]**

Enables Group 1 interrupts for the Non-secure state.
**EnableGrp1NS** | **Meaning**
--- | ---
0b0 | Non-secure Group 1 interrupts are disabled.
0b1 | Non-secure Group 1 interrupts are enabled.

The Non-secure `ICC_IGRPEN1_EL1`.Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

This field resets to 0.

## Accessing the ICC_IGRPEN1_EL3

Accesses to this register use the following encodings:

**MRS <Xt>, ICC_IGRPEN1_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN1_EL3;
end

**MSR ICC_IGRPEN1_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN1_EL3 = X[t];
ICC_PMR_EL1, Interrupt Controller Interrupt Priority Mask Register

The ICC_PMR_EL1 characteristics are:

**Purpose**

Provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Writes to this register must be high performance and must ensure that no interrupt of lower priority than the written value occurs after the write, without requiring an ISB or an exception boundary.

**Configuration**

AArch64 System register ICC_PMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICC_PMR[31:0].

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. See Observability of the effects of accesses to the GIC registers, for more information.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_PMR_EL1 is a 64-bit register.

**Field descriptions**

The ICC_PMR_EL1 bit assignments are:

```
  63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32
  |       RES0       | Priority
  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

**Bits [63:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

The possible priority field values are as follows:
### Implemented priority bits

<table>
<thead>
<tr>
<th>[7:0]</th>
<th>Possible priority field values</th>
<th>Number of priority levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:0]</td>
<td>0x00-0xFF (0-255), all values</td>
<td>256</td>
</tr>
<tr>
<td>[7:1]</td>
<td>0x00-0xFE (0-254), even values only</td>
<td>128</td>
</tr>
<tr>
<td>[7:2]</td>
<td>0x00-0xFC (0-252), in steps of 4</td>
<td>64</td>
</tr>
<tr>
<td>[7:3]</td>
<td>0x00-0xF8 (0-248), in steps of 8</td>
<td>32</td>
</tr>
<tr>
<td>[7:4]</td>
<td>0x00-0xF0 (0-240), in steps of 16</td>
<td>16</td>
</tr>
</tbody>
</table>

Unimplemented priority bits are RAZ/WI.

This field resets to 0.

## Accessing the ICC_PMR_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, ICC_PMR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
elsalif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsalif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_PMR_EL1;
elsalif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_PMR_EL1;
elsalif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ICC_PMR_EL1;
elsalif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsalif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ICC_PMR_EL1;
elsalif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return ICC_PMR_EL1;
```

**MSR ICC_PMR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then  
  UNDEFINED; 
elsif PSTATE.EL == EL1 then  
  if ICC_SRE_EL1.SRE == '0' then 
    AArch64.SystemAccessTrap(EL1, 0x18); 
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then 
    AArch64.SystemAccessTrap(EL2, 0x18); 
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then 
    ICC_PMR_EL1 = X[t]; 
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then 
    ICC_PMR_EL1 = X[t]; 
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then 
    AArch64.SystemAccessTrap(EL3, 0x18); 
  else 
    ICC_PMR_EL1 = X[t]; 
  endif 
elsif PSTATE.EL == EL2 then  
  if ICC_SRE_EL2.SRE == '0' then 
    AArch64.SystemAccessTrap(EL2, 0x18); 
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then 
    AArch64.SystemAccessTrap(EL3, 0x18); 
  else 
    ICC_PMR_EL1 = X[t]; 
  endif 
elsif PSTATE.EL == EL3 then  
  if ICC_SRE_EL3.SRE == '0' then 
    AArch64.SystemAccessTrap(EL3, 0x18); 
  else 
    ICC_PMR_EL1 = X[t]; 
  endif

ICC_RPR_EL1, Interrupt Controller Running Priority Register

The ICC_RPR_EL1 characteristics are:

**Purpose**

Indicates the Running priority of the CPU interface.

**Configuration**

AArch64 System register ICC_RPR_EL1 performs the same function as AArch32 System register ICC_RPR.

**Attributes**

ICC_RPR_EL1 is a 64-bit register.

**Field descriptions**

The ICC_RPR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:8]</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>[7:0]</td>
<td>Priority</td>
</tr>
</tbody>
</table>

**Bits [63:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

**Note**

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

**Accessing the ICC_RPR_EL1**

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings:

MRS <Xt>, ICC_RPR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICC_RPR_EL1;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IM0 == '1' then
    return ICC_RPR_EL1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_RPR_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_RPR_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_RPR_EL1;
The ICC_SGI0R_EL1 characteristics are:

**Purpose**

Generates Secure Group 0 SGIs.

**Configuration**

AArch64 System register ICC_SGI0R_EL1 performs the same function as AArch32 System register ICC_SGI0R.

**Attributes**

ICC_SGI0R_EL1 is a 64-bit register.

**Field descriptions**

The ICC_SGI0R_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Aff3</td>
<td>INTID</td>
<td>RES0</td>
<td>Aff1</td>
<td>RS</td>
<td>RES0</td>
<td>IRM</td>
<td>Aff2</td>
<td>TargetList</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.
Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

<table>
<thead>
<tr>
<th>IRM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.&lt;target list&gt;.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to all PEs in the system, excluding “self”.</td>
</tr>
</tbody>
</table>

**Aff2, bits [39:32]**

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**Bits [31:28]**

Reserved, RES0.

**INTID, bits [27:24]**

The INTID of the SGI.

**Aff1, bits [23:16]**

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**TargetList, bits [15:0]**

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

---

**Note**

This restricts a system to sending targeted SGIs to PEs with an affinity 0 number that is less than 16.

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

---

**Accessing the ICC_SGI0R_EL1**

This register allows software executing in a Secure state to generate Group 0 SGIs. It will also allow software executing in a Non-secure state to generate Group 0 SGIs, if permitted by the settings of GICR_NSACR in the Redistributor corresponding to the target PE.

When GICD_CTRLR.DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the GICR_NSACR register associated with the target PE. For more information see Use of control registers for SGI forwarding.

---

**Note**
Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings:

MSR ICC_SGI0R_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI0R_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI0R_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SGI0R_EL1 = X[t];
    endif
The ICC_SGI1R_EL1 characteristics are:

**Purpose**
Generates Group 1 SGIs for the current Security state.

**Configuration**
AArch64 System register ICC_SGI1R_EL1 performs the same function as AArch32 System register ICC_SGI1R.
Under certain conditions a write to ICC_SGI1R_EL1 can generate Group 0 interrupts, see Forwarding an SGI to a target PE.

**Attributes**
ICC_SGI1R_EL1 is a 64-bit register.

**Field descriptions**
The ICC_SGI1R_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Affinity 3 (Aff3)</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>RangeSelector (RS)</td>
</tr>
<tr>
<td>31</td>
<td>TargetList</td>
</tr>
</tbody>
</table>

**Bits [63:56]**
Reserved, RES0.

**Aff3, bits [55:48]**
The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.
If the IRM bit is 1, this field is RES0.

**RS, bits [47:44]**
RangeSelector
Controls which group of 16 values is represented by the TargetList field.
TargetList[n] represents aff0 value ((RS * 16) + n).
When ICC_CTLR_EL1.RSS==0, RS is RES0.
When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a **CONSTRAINED UNPREDICTABLE** choice of:

- The write is ignored.
- The RS field is treated as 0.
Bits [43:41]
Reserved, RES0.

IRM, bit [40]
Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

<table>
<thead>
<tr>
<th>IRM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.&lt;target list&gt;.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to all PEs in the system, excluding &quot;self&quot;.</td>
</tr>
</tbody>
</table>

Aff2, bits [39:32]
The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.
If the IRM bit is 1, this field is RES0.

Bits [31:28]
Reserved, RES0.

INTID, bits [27:24]
The INTID of the SGI.

Aff1, bits [23:16]
The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.
If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]
Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.
If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note
This restricts a system to sending targeted SGIs to PEs with an affinity 0 number that is less than 16.
If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.
If the IRM bit is 1, this field is RES0.

Accessing the ICC_SGI1R_EL1

Note
Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.
Accesses to this register use the following encodings:

**MSR ICC_SGI1R_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_SGI1R_EL1 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_SGI1R_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_SGI1R_EL1 = X[t];
  endif
ICC_SRE_EL1, Interrupt Controller System Register Enable register (EL1)

The ICC_SRE_EL1 characteristics are:

### Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL1.

### Configuration

AArch64 System register ICC_SRE_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register ICC_SRE[31:0] (S).

AArch64 System register ICC_SRE_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register ICC_SRE[31:0] (NS).

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

ICC_SRE_EL1 is a 64-bit register.

### Field descriptions

The ICC_SRE_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>DIB</td>
</tr>
<tr>
<td>60</td>
<td>DF</td>
</tr>
<tr>
<td>59</td>
<td>SRE</td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:3]**

Reserved, RES0.

**DIB, bit [2]**

Disable IRQ bypass.

<table>
<thead>
<tr>
<th>DIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and \texttt{GICD_CTLR.DS} == 0, this field is a read-only alias of \texttt{ICC_SRE_EL3.DIB}.

If EL3 is implemented and \texttt{GICD_CTLR.DS} == 1, and EL2 is not implemented, this field is a read-write alias of \texttt{ICC_SRE_EL3.DIB}.

If EL3 is not implemented or \texttt{GICD_CTLR.DS} == 1, and EL2 is implemented, this field is a read-only alias of \texttt{ICC_SRE_EL2.DIB}.

In systems that do not support IRQ bypass, this field is RAO/WI.

This field resets to 0.
DFB, bit [1]

Disable FIQ bypass.

<table>
<thead>
<tr>
<th>DFB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and `GICD_CTRL.DS == 0`, this field is a read-only alias of `ICC_SRE_EL3.DFB`.

If EL3 is implemented and `GICD_CTRL.DS == 1`, and EL2 is not implemented, this field is a read-write alias of `ICC_SRE_EL3.DFB`.

If EL3 is not implemented or `GICD_CTRL.DS == 1`, and EL2 is implemented, this field is a read-only alias of `ICC_SRE_EL2.DFB`.

In systems that do not support FIQ bypass, this field is RAO/WI.

This field resets to 0.

SRE, bit [0]

System Register Enable.

<table>
<thead>
<tr>
<th>SRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The memory-mapped interface must be used. Access at EL1 to any ICC_* System register other than ICC_SRE_EL1 is trapped to EL1</td>
</tr>
<tr>
<td>0b1</td>
<td>The System register interface for the current Security state is enabled.</td>
</tr>
</tbody>
</table>

If software changes this bit from 1 to 0 in the Secure instance of this register, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and `ICC_SRE_EL3.SRE==0` the Secure copy of this bit is RAZ/WI. If `ICC_SRE_EL3.SRE` is changed from zero to one, the Secure copy of this bit becomes UNKNOWN.

If EL2 is implemented and `ICC_SRE_EL2.SRE==0` the Non-secure copy of this bit is RAZ/WI. If `ICC_SRE_EL2.SRE` is changed from zero to one, the Non-secure copy of this bit becomes UNKNOWN.

If EL3 is implemented and `ICC_SRE_EL3.SRE==0` the Non-secure copy of this bit is RAZ/WI. If `ICC_SRE_EL3.SRE` is changed from zero to one, the Non-secure copy of this bit becomes UNKNOWN.

GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI. The following options are supported:

- The Non-secure copy of `ICC_SRE_EL1.SRE` can be RAO/WI if `ICC_SRE_EL2.SRE` is also RAO/WI. This means all Non-secure software, including VMs using only virtual interrupts, must access the GIC using System registers.
- The Secure copy of `ICC_SRE_EL1.SRE` can be RAO/WI if `ICC_SRE_EL3.SRE` and `ICC_SRE_EL2.SRE` are also RAO/WI. This means that all Secure software must access the GIC using System registers and all Non-secure accesses to registers for physical interrupts must use System registers.

**Note**

A VM using only virtual interrupts might still use memory-mapped access if the Non-secure copy of `ICC_SRE_EL1.SRE` is not RAO/WI.

This field resets to 0.

**Accessing the ICC_SRE_EL1**

Execution with `ICC_SRE_EL1.SRE` set to 0 might make some System registers UNKNOWN.

Accesses to this register use the following encodings:
MRS <Xt>, ICC_SRE_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && ICC_SRE_EL2.Enable == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      return ICC_SRE_EL1_S;
    else
      return ICC_SRE_EL1_NS;
  else
    return ICC_SRE_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      return ICC_SRE_EL1_S;
    else
      return ICC_SRE_EL1_NS;
  else
    return ICC_SRE_EL1;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    return ICC_SRE_EL1_S;
  else
    return ICC_SRE_EL1_NS;
else
  return ICC_SRE_EL1;

MSR ICC_SRE_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

ICC_SRE_EL1, Interrupt Controller System Register Enable register (EL1)
if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && ICC_SRE_EL2.Enable == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elif HaveEL(EL3) then
  if SCR_EL3.NS == '0' then
    ICC_SRE_EL1_S = X[t];
  else
    ICC_SRE_EL1_NS = X[t];
else
  ICC_SRE_EL1 = X[t];
elif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elif HaveEL(EL3) then
  if SCR_EL3.NS == '0' then
    ICC_SRE_EL1_S = X[t];
  else
    ICC_SRE_EL1_NS = X[t];
else
  ICC_SRE_EL1 = X[t];
elif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    ICC_SRE_EL1_S = X[t];
else
  ICC_SRE_EL1_NS = X[t];
ICC_SRE_EL2, Interrupt Controller System Register Enable register (EL2)

The ICC_SRE_EL2 characteristics are:

**Purpose**

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

**Configuration**

AArch64 System register ICC_SRE_EL2 is architecturally mapped to AArch32 System register ICC_HSRE.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_SRE_EL2 is a 64-bit register.

**Field descriptions**

The ICC_SRE_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>Enable</td>
<td>Enables lower Exception level access to ICC_SRE_EL1.</td>
</tr>
<tr>
<td>2</td>
<td>DIB</td>
<td>Disable IRQ bypass.</td>
</tr>
</tbody>
</table>

**Bits [63:4]**

Reserved, RES0.

**Enable, bit [3]**

Enable. Enables lower Exception level access to ICC_SRE_EL1.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When EL2 is implemented and enabled in the current Security state, EL1 accesses to ICC_SRE_EL1 trap to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to ICC_SRE_EL1 do not trap to EL2.</td>
</tr>
</tbody>
</table>

If ICC_SRE_EL2.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_SRE_EL2.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

This field resets to an architecturally UNKNOWN value.

**DIB, bit [2]**

Disable IRQ bypass.
DIB, bit [0]

Disable IRQ bypass.

<table>
<thead>
<tr>
<th>DIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and \( \text{GICD_CTLR.DS} \) is 0, this field is a read-only alias of \( \text{ICC_SRE_EL3.DIB} \).

If EL3 is implemented and \( \text{GICD_CTLR.DS} \) is 1, this field is a read-write alias of \( \text{ICC_SRE_EL3.DIB} \).

In systems that do not support IRQ bypass, this bit is RAO/WI.

This field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

<table>
<thead>
<tr>
<th>DFB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and \( \text{GICD_CTLR.DS} \) is 0, this field is a read-only alias of \( \text{ICC_SRE_EL3.DFB} \).

If EL3 is implemented and \( \text{GICD_CTLR.DS} \) is 1, this field is a read-write alias of \( \text{ICC_SRE_EL3.DFB} \).

In systems that do not support FIQ bypass, this bit is RAO/WI.

This field resets to 0.

SRE, bit [0]

System Register Enable.

<table>
<thead>
<tr>
<th>SRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The memory-mapped interface must be used. Access at EL2 to any ICH_* or ICC_* register other than ( \text{ICC_SRE_EL1} ) or ( \text{ICC_SRE_EL2} ), is trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>The System register interface to the ICH_* registers and the EL1 and EL2 ICC_* registers is enabled for EL2.</td>
</tr>
</tbody>
</table>

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and \( \text{ICC_SRE_EL3.SRE} \) == 0 this bit is RAZ/WI. If \( \text{ICC_SRE_EL3.SRE} \) is changed from zero to one, this bit becomes UNKNOWN.

GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI, but this is only allowed if \( \text{ICC_SRE_EL3.SRE} \) is also RAO/WI.

This field resets to 0.

Accessing the ICC_SRE_EL2

Execution with \( \text{ICC_SRE_EL2.SRE} \) set to 0 might make some System registers UNKNOWN.

Accesses to this register use the following encodings:

\[ \text{MRS } <Xt>, \text{ ICC_SRE_EL2} \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_SRE_EL2;
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        return ICC_SRE_EL2;

MSR ICC_SRE_EL2, \langle Xt\rangle

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SRE_EL2 = \langle Xt\rangle;
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        ICC_SRE_EL2 = \langle Xt\rangle;
ICC_SRE_EL3, Interrupt Controller System Register Enable register (EL3)

The ICC_SRE_EL3 characteristics are:

**Purpose**

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

**Configuration**

AArch64 System register ICC_SRE_EL3 bits [31:0] can be mapped to AArch32 System register ICC_MSRE[31:0], but this is not architecturally mandated.

This register is present only when HaveEL(EL3). Otherwise, direct accesses to ICC_SRE_EL3 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_SRE_EL3 is a 64-bit register.

**Field descriptions**

The ICC_SRE_EL3 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | RES0 | Enable | DIB | DBSRE |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**Bits [63:4]**

Reserved, RES0.

**Enable, bit [3]**

Enable. Enables lower Exception level access to ICC_SRE_EL1 and ICC_SRE_EL2.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1 accesses to ICC_SRE_EL1 trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_SRE_EL2.Enable == 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to ICC_SRE_EL1 trap to EL3. EL2 accesses to ICC_SRE_EL1 and ICC_SRE_EL2 do not trap to EL3.</td>
</tr>
</tbody>
</table>

If ICC_SRE_EL3.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_SRE_EL3.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

This field resets to an architecturally UNKNOWN value.
DIB, bit [2]

Disable IRQ bypass.

<table>
<thead>
<tr>
<th>DIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQ bypass disabled.</td>
</tr>
</tbody>
</table>

In systems that do not support IRQ bypass, this bit is RAO/WI.

This field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

<table>
<thead>
<tr>
<th>DFB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQ bypass disabled.</td>
</tr>
</tbody>
</table>

In systems that do not support FIQ bypass, this bit is RAO/WI.

This field resets to 0.

SRE, bit [0]

System Register Enable.

<table>
<thead>
<tr>
<th>SRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The memory-mapped interface must be used. Access at EL3 to any ICH_* or ICC_* register other than ICC_SRE_EL1, ICC_SRE_EL2, or ICC_SRE_EL3 is trapped to EL3</td>
</tr>
<tr>
<td>0b1</td>
<td>The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3</td>
</tr>
</tbody>
</table>

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI.

This field resets to 0.

**Accessing the ICC_SRE_EL3**

This register is always System register accessible.

Accesses to this register use the following encodings:

MRS <Xt>, ICC_SRE_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b10</td>
<td>0b100</td>
<td>0b100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ICC_SRE_EL3;
MSR ICC_SRE_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  ICC_SRE_EL3 = X[t];
The ICH_AP0R<n>_EL2 characteristics are:

**Purpose**

Provides information about Group 0 virtual active priorities for EL2.

**Configuration**

AArch64 System register ICH_AP0R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_AP0R<n>[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_AP0R<n>_EL2 is a 64-bit register.

**Field descriptions**

The ICH_AP0R<n>_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:32]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>P&lt;x&gt;</strong></td>
<td>There is no Group 0 interrupt active with this priority level, or all active Group 0 interrupts with this priority level have undergone priority-drop.</td>
</tr>
<tr>
<td>0b0</td>
<td>There is a Group 0 interrupt active with this priority level which has not undergone priority drop.</td>
</tr>
<tr>
<td>0b1</td>
<td>There is a Group 0 interrupt active with this priority level which has not undergone priority drop.</td>
</tr>
</tbody>
</table>

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0_EL2 in the bits corresponding to 0:Priority[6:2].
The active state of preemption levels 128 - 252 are held in ICH_AP0R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0_EL2 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP0R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP0R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

This field resets to 0.

Software must ensure that ICH_AP0R<n>_EL2 is 0 for legacy VMs otherwise behaviour is UNPREDICTABLE. For more information about support for legacy VMs, see Support for legacy operation of VMs.

The active priorities for Group 0 and Group 1 interrupts for legacy VMs are held in ICH_AP1R<n>_EL2 and reads and writes to GICV_APR access ICH_AP1R<n>_EL2. This means that ICH_AP0R<n>_EL2 is inaccessible to legacy VMs.

### Accessing the ICH_AP0R<n>_EL2

ICH_AP0R_EL2 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP0R2_EL2 and ICH_AP0R3_EL2 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

- Virtual interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution at EL1 or EL0.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>_EL2.
- ICH_AP1R<n>_EL2.

Having the bit corresponding to a priority set in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 can result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x480+8*UInt(op2<1:0>)];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ICH_AP0R_EL2[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICH_AP0R_EL2[UInt(op2<1:0>)];

MSR ICH_AP0R<n>_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x480+8*UInt(op2<1:0>)] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    ICH_AP0R_EL2[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICH_AP0R_EL2[UInt(op2<1:0>)] = X[t];
ICH_AP1R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n>_EL2 characteristics are:

**Purpose**

Provides information about Group 1 virtual active priorities for EL2.

**Configuration**

AArch64 System register ICH_AP1R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_AP1R<n>[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_AP1R<n>_EL2 is a 64-bit register.

**Field descriptions**

The ICH_AP1R<n>_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**Bits [63:32]**

Reserved, RES0.

**P<x>, bit [x], for x = 0 to 31**

Group 1 interrupt active priorities. Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no Group 1 interrupt active with this priority level, or all active Group 1 interrupts with this priority level have undergone priority-drop.</td>
</tr>
<tr>
<td>0b1</td>
<td>There is a Group 1 interrupt active with this priority level which has not undergone priority drop.</td>
</tr>
</tbody>
</table>

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0_EL2 in the bits corresponding to Priority[6:2].
• The active state of preemption levels 128 - 252 are held in ICH_AP1R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

• The active state of preemption levels 0 - 62 are held in ICH_AP1R0_EL2 in the bits corresponding to 00:Priority[5:1].
• The active state of preemption levels 64 - 126 are held in ICH_AP1R1_EL2 in the bits corresponding to 01:Priority[5:1].
• The active state of preemption levels 128 - 190 are held in ICH_AP1R2_EL2 in the bits corresponding to 10:Priority[5:1].
• The active state of preemption levels 192 - 254 are held in ICH_AP1R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

This field resets to 0.

This register is always used for legacy VMs, regardless of the group of the virtual interrupt. Reads and writes to GICV_APR<n> access ICH_AP1R<n>_EL2. For more information about support for legacy VMs, see Support for legacy operation of VMs.

Accessing the ICH_AP1R<n>_EL2

ICH_AP1R_EL2 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP1R2_EL2 and ICH_AP1R3_EL2 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by ICH_VTR_EL2.PREbits.

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

Accesses to this register use the following encodings:

MRS <Xt>, ICH_AP1R<n>_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x4A0+8*UInt(op2<1:0>)];
  elif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ICH_AP1R_EL2[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICH_AP1R_EL2[UInt(op2<1:0>)];

MSR ICH_AP1R<n>_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x4A0+8*UInt(op2<1:0>)] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    ICH_AP1R_EL2[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICH_AP1R_EL2[UInt(op2<1:0>)] = X[t];

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ICH_EISR_EL2, Interrupt Controller End of Interrupt Status Register

The ICH_EISR_EL2 characteristics are:

**Purpose**

Indicates which List registers have outstanding EOI maintenance interrupts.

**Configuration**

AArch64 System register ICH_EISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_EISR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

**Attributes**

ICH_EISR_EL2 is a 64-bit register.

**Field descriptions**

The ICH_EISR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Status&lt;n&gt;, bit [n], for n = 0 to 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 0 to 15**

EOI maintenance interrupt status bit for List register <n>:

<table>
<thead>
<tr>
<th>Status&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>List register &lt;n&gt;, ICH_LR&lt;n&gt;_EL2, does not have an EOI maintenance interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>List register &lt;n&gt;, ICH_LR&lt;n&gt;_EL2, has an EOI maintenance interrupt that has not been handled.</td>
</tr>
</tbody>
</table>

For any ICH_LR<n>_EL2, the corresponding status bit is set to 1 if all of the following are true:

- ICH_LR<n>_EL2 State is 0b00.
- ICH_LR<n>_EL2 HW is 0.
- ICH_LR<n>_EL2 EOI (bit [41]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

Otherwise the status bit takes the value 0.

**Accessing the ICH_EISR_EL2**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_EISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_EISR_EL2;

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
</tr>
</tbody>
</table>
ICH_ELRSR_EL2, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR_EL2 characteristics are:

**Purpose**

These registers can be used to locate a usable List register when the hypervisor is delivering an interrupt to a VM.

**Configuration**

AArch64 System register ICH_ELRSR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_ELRSR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

**Attributes**

ICH_ELRSR_EL2 is a 64-bit register.

**Field descriptions**

The ICH_ELRSR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Status<n>, bit [n], for n = 0 to 15 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 0 to 15**

Status bit for List register <n>, ICH_LR<n>_EL2:

<table>
<thead>
<tr>
<th>Status&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>List register ICH_LR&lt;n&gt;_EL2, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>List register ICH_LR&lt;n&gt;_EL2 does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt.</td>
</tr>
</tbody>
</table>

For any List register <n>, the corresponding status bit is set to 1 if ICH_LR<n>_EL2 State is 0b00 and either ICH_LR<n>_EL2.HW is 1 or ICH_LR<n>_EL2.EOI (bit [41]) is 0.

Otherwise the status bit takes the value 0.

**Accessing the ICH_ELRSR_EL2**

Accesses to this register use the following encodings:
MRS <Xt>, ICH_ELRSR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ICH_ELRSR_EL2;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICH_ELRSR_EL2;
ICH_HCR_EL2, Interrupt Controller Hyp Control Register

The ICH_HCR_EL2 characteristics are:

**Purpose**

Controls the environment for VMs.

**Configuration**

AArch64 System register ICH_HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register `ICH_HCR[31:0]`.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally *UNKNOWN* values.

**Attributes**

ICH_HCR_EL2 is a 64-bit register.

**Field descriptions**

The ICH_HCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:32]</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOIcount</td>
</tr>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40</td>
</tr>
<tr>
<td>632616059585756555453525150494847</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**EOIcount, bits [31:27]**

This field is incremented whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is **CONSTRAINED UNPREDICTABLE** whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers (`ICH_AP0R<n>_EL2/ICH_AP1R<n>_EL2`) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

This field resets to 0.
Bits [26:15]

Reserved, RES0.

**TDIR, bit [14]**

Trap EL1 writes to **ICC_DIR_EL1** and **ICV_DIR_EL1**.

<table>
<thead>
<tr>
<th>TDIR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1 writes of <strong>ICC_DIR_EL1</strong> and <strong>ICV_DIR_EL1</strong> are not trapped to EL2, unless trapped by other mechanisms.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 writes of <strong>ICV_DIR_EL1</strong> are trapped to EL2. It is IMPLEMENTATION DEFINED whether writes of <strong>ICC_DIR_EL1</strong> are trapped. Not trapping <strong>ICC_DIR_EL1</strong> writes is DEPRECATED.</td>
</tr>
</tbody>
</table>

Support for this bit is OPTIONAL, with support indicated by **ICH_VTR_EL2**.

If the implementation does not support this trap, this bit is RES0.

Arm deprecates not including this trap bit.

This field resets to 0.

**TSEI, bit [13]**

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at EL1.

<table>
<thead>
<tr>
<th>TSEI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Locally generated SEIs do not cause a trap to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Locally generated SEIs trap to EL2.</td>
</tr>
</tbody>
</table>

If **ICH_VTR_EL2**.SEIS is 0, this bit is RES0.

This field resets to 0.

**TALL1, bit [12]**

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

<table>
<thead>
<tr>
<th>TALL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**TALL0, bit [11]**

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

<table>
<thead>
<tr>
<th>TALL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**TC, bit [10]**

Trap all EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.
<table>
<thead>
<tr>
<th>TC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1 accesses to common registers proceed as normal.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 accesses to common registers trap to EL2.</td>
</tr>
</tbody>
</table>

This affects accesses to ICC_SGI0R_EL1, ICC_SGI1R_EL1, ICC_ASGI1R_EL1, ICC_CTLR_EL1, ICC_DIR_EL1, ICC_PMR_EL1, ICC_RPR_EL1, ICC_CTLR_EL1, ICC_DIR_EL1, ICC_PMR_EL1, and ICC_RPR_EL1.

This field resets to 0.

**Bits [9:8]**

Reserved, RES0.

**VGrp1DIE, bit [7]**

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

<table>
<thead>
<tr>
<th>VGrp1DIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**VGrp1EIE, bit [6]**

VM Group 1 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

<table>
<thead>
<tr>
<th>VGrp1EIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**VGrp0DIE, bit [5]**

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

<table>
<thead>
<tr>
<th>VGrp0DIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**VGrp0EIE, bit [4]**

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

<table>
<thead>
<tr>
<th>VGrp0EIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.
NPIE, bit [3]

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

<table>
<thead>
<tr>
<th>NPIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.</td>
</tr>
</tbody>
</table>

This field resets to 0.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

<table>
<thead>
<tr>
<th>LRENPIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt is asserted while the EOICount field is not 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

UIE, bit [1]

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

<table>
<thead>
<tr>
<th>UIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.</td>
</tr>
</tbody>
</table>

This field resets to 0.

En, bit [0]

Enable. Global enable bit for the virtual CPU interface:

<table>
<thead>
<tr>
<th>En</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual CPU interface operation disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual CPU interface operation enabled.</td>
</tr>
</tbody>
</table>

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of ICV_IAR0_EL1, ICV_IAR1_EL1, GICV_IAR or GICV_AIAR returns a spurious interrupt ID.

Note

This field is RES0 when SCR_EL3.{NS,EEL2} == {0,0}

This field resets to 0.

Accessing the ICH_HCR_EL2

Accesses to this register use the following encodings:

```
MRS <Xt>, ICH_HCR_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>
ICH_HCR_EL2, Interrupt Controller Hyp Control Register

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x4C0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
else
    PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_HCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_HCR_EL2;

MSR ICH_HCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x4C0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
else
    PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_HCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_HCR_EL2 = X[t];
ICH_LR<n>_EL2, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n>_EL2 characteristics are:

**Purpose**

Provides interrupt context information for the virtual CPU interface.

**Configuration**

AArch64 System register ICH_LR<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_LR<n>[31:0].

AArch64 System register ICH_LR<n>_EL2 bits [63:32] are architecturally mapped to AArch32 System register ICH_LRC<n>[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_LR<n>_EL2 is a 64-bit register.

**Field descriptions**

The ICH_LR<n>_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
<td>The state of the interrupt:</td>
</tr>
<tr>
<td>HW Group</td>
<td>RES0</td>
</tr>
<tr>
<td>vINTID</td>
<td></td>
</tr>
<tr>
<td>31-0</td>
<td>27-26</td>
</tr>
</tbody>
</table>

**State, bits [63:62]**

The state of the interrupt:

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Invalid (Inactive).</td>
</tr>
<tr>
<td>0b01</td>
<td>Pending.</td>
</tr>
<tr>
<td>0b10</td>
<td>Active.</td>
</tr>
<tr>
<td>0b11</td>
<td>Pending and active.</td>
</tr>
</tbody>
</table>

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

This field resets to an architecturally UNKNOWN value.
HW, bit [61]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the ID that the pINTID field indicates.

<table>
<thead>
<tr>
<th>HW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated.</td>
</tr>
<tr>
<td>0b1</td>
<td>The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical interrupt ID.</td>
</tr>
</tbody>
</table>

If ICH_VMCR_EL2.VEOIM is 0, this request corresponds to a write to ICC_EOIR0_EL1 or ICC_EOIR1_EL1. Otherwise, it corresponds to a write to ICC_DIR_EL1.

This field resets to an architecturally UNKNOWN value.

Group, bit [60]

Indicates the group for this virtual interrupt.

<table>
<thead>
<tr>
<th>Group</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This is a Group 0 virtual interrupt. ICH_VMCR_EL2.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR_EL2.VENG0 enables signaling of this interrupt to the virtual machine.</td>
</tr>
<tr>
<td>0b1</td>
<td>This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR_EL2.VENG1 enables the signalling of this interrupt to the virtual machine. If ICH_VMCR_EL2.VCBPR is 0, then ICC_BPR1_EL1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR&lt;n&gt;_EL2 determines preemption.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bits [59:56]

Reserved, RES0.

Priority, bits [55:48]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[48] up to bit[50]. The number of implemented bits can be discovered from ICH_VTR_EL2.PRIBits.

This field resets to an architecturally UNKNOWN value.

Bits [47:45]

Reserved, RES0.

pINTID, bits [44:32]

Physical INTID, for hardware interrupts.

When ICH_LR<n>_EL2.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[44:42]: RES0.
- Bit[41]: EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, a maintenance interrupt is asserted.
• Bits[40:32] : RES0.

When ICH_LR<n>_EL2.HW is 1 (there is a corresponding physical interrupt):

• This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
• When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
• If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR_EL1.IDbits.

This field resets to an architecturally UNKNOWN value.

**vINTID, bits [31:0]**

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LR<n>_EL2.State!=0b00 (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

• ICH_LR<n>_EL2.State == 0b01.
• ICH_LR<n>_EL2.State == 0b10.
• ICH_LR<n>_EL2.State == 0b11.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICC_VTR_EL2.IDbits.

When ICC_SRE_EL1.SRE == 0, specifying a vINTID in the LPI range is UNPREDICTABLE.

---

**Note**

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

This field resets to an architecturally UNKNOWN value.

**Accessing the ICH_LR<n>_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, ICH_LR<n>_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b110:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x400+8*UInt(CRm<0>:op2<2:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
end if;
ICH_MISR_EL2, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR_EL2 characteristics are:

**Purpose**

Indicates which maintenance interrupts are asserted.

**Configuration**

AArch64 System register ICH_MISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_MISR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_MISR_EL2 is a 64-bit register.

**Field descriptions**

The ICH_MISR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>VGrp1D</th>
<th>VGrp1E</th>
<th>VGrp0D</th>
<th>VGrp0E</th>
<th>NP</th>
<th>LRENP</th>
<th>U</th>
<th>EOI</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:8]**

Reserved, RES0.

**VGrp1D, bit [7]**

vPE Group 1 Disabled.

<table>
<thead>
<tr>
<th>VGrp1D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 1 Disabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 1 Disabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when ICH_HCR_EL2.VGrp1DIE==1 and ICH_VMCR_EL2.VENG1==is 0.

This field resets to 0.

**VGrp1E, bit [6]**

vPE Group 1 Enabled.

<table>
<thead>
<tr>
<th>VGrp1E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 1 Enabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 1 Enabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>
This maintenance interrupt is asserted when `ICH_HCR_EL2.VGrp1EIE==1` and `ICH_VMCR_EL2.VENG1==is 1.

This field resets to 0.

**VGrp0D, bit [5]**

vPE Group 0 Disabled.

<table>
<thead>
<tr>
<th>VGrp0D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 0 Disabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 0 Disabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when `ICH_HCR_EL2.VGrp0DIE==1` and `ICH_VMCR_EL2.VENG0==is 0.

This field resets to 0.

**VGrp0E, bit [4]**

vPE Group 0 Enabled.

<table>
<thead>
<tr>
<th>VGrp0E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 0 Enabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 0 Enabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when `ICH_HCR_EL2.VGrp0EIE==1` and `ICH_VMCR_EL2.VENG0==1.0.

This field resets to 0.

**NP, bit [3]**

No Pending.

<table>
<thead>
<tr>
<th>NP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Pending maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>No Pending maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when `ICH_HCR_EL2.NPIE==1` and no List register is in pending state.

This field resets to 0.

**LRENP, bit [2]**

List Register Entry Not Present.

<table>
<thead>
<tr>
<th>LRENP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>List Register Entry Not Present maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>List Register Entry Not Present maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when `ICH_HCR_EL2.LRENPIE==1` and `ICH_HCR_EL2.EOICount is non-zero. This field resets to 0.

**U, bit [1]**

Underflow.

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when `ICH_HCR_EL2.UIE==1` and zero or one of the List register entries are marked as a valid interrupt, that is, if the corresponding `ICH_LR<n>_EL2.State` bits do not equal 0x0.
This field resets to 0.

**EOI, bit [0]**

End Of Interrupt.

<table>
<thead>
<tr>
<th>EOI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>End Of Interrupt maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>End Of Interrupt maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when at least one bit in **ICH_EISR_EL2** is 1.

This field resets to 0.

The U and NP bits do not include the status of any pending/active VSET packets because these bits control generation of interrupts that allow software management of the contents of the List Registers (which are not affected by VSET packets).

**Accessing the ICH_MISR_EL2**

Accesses to this register use the following encodings:

```
MRS <Xt>, ICH_MISR_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ICH_MISR_EL2;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICH_MISR_EL2;
ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR_EL2 characteristics are:

**Purpose**

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

**Configuration**

AArch64 System register ICH_VMCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_VMCR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

**Attributes**

ICH_VMCR_EL2 is a 64-bit register.

**Field descriptions**

The ICH_VMCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>Virtual Priority Mask (VPMR)</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Virtual Binary Point Register 0 (VBPR0)</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Virtual Binary Point Register 1 (VBPR1)</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
<tr>
<td>58</td>
<td>Virtual Enable Interrupt Mask (VEOIM)</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
<tr>
<td>56</td>
<td>Virtual CASR Interval (VCBPR)</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Virtual FiQ Enable (VFIQEn)</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Virtual Acknowledge Control (VAckCtl)</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Virtual Enable Group 1 (VENG1)</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>Virtual Enable Group 0 (VENG0)</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**VPMR, bits [31:24]**

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of ICH_VPMR_EL2.Priority.

**VBPR0, bits [23:21]**

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR_EL2.VCBPR == 1.

This field is an alias of ICH_BPR0_EL1.BinaryPoint.

The minimum value of this field is determined by ICH_VTR_EL2.PREbits. An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.
**VBPR1, bits [20:18]**

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if `ICH_VMCR_EL2.VCBPR == 0`.

This field is an alias of `ICV_BPR1_EL1.BinaryPoint`.

This field is always accessible to EL2 accesses, regardless of the setting of the `ICH_VMCR_EL2.VCBPR` field.

For Non-secure writes, the minimum value of this field is the minimum value of `ICH_VMCR_EL2.VBPR0` plus one.

For Secure writes, the minimum value of this field is the minimum value of `ICH_VMCR_EL2.VBPR0`.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

**Bits [17:10]**

Reserved, RES0.

**VEOIM, bit [9]**

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

<table>
<thead>
<tr>
<th>VEOIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><code>ICV_EOIR0_EL1</code> and <code>ICV_EOIR1_EL1</code> provide both priority drop and interrupt deactivation functionality. Accesses to <code>ICV_DIR_EL1</code> are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td><code>ICV_EOIR0_EL1</code> and <code>ICV_EOIR1_EL1</code> provide priority drop functionality only. <code>ICV_DIR_EL1</code> provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This bit is an alias of `ICV_CTLR_EL1.EOImode`.

**Bits [8:5]**

Reserved, RES0.

**VCBPR, bit [4]**

Virtual Common Binary Point Register. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VCBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><code>ICV_BPR1_EL1</code> determines the preemption group for virtual Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reads of <code>ICV_BPR1_EL1</code> return <code>ICV_BPR0_EL1</code> plus one, saturated to 0b111. Writes to <code>ICV_BPR1_EL1</code> are ignored.</td>
</tr>
</tbody>
</table>

This field is an alias of `ICV_CTLR_EL1.CBPR`.

**VFIQEn, bit [3]**

Virtual FIQ enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VFIQEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 virtual interrupts are presented as virtual IRQs.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 virtual interrupts are presented as virtual FIQs.</td>
</tr>
</tbody>
</table>

This bit is an alias of `GICV_CTLR.FIQEn`.

In implementations where the Non-secure copy of `ICC_SRE_EL1` SRE is always 1, this bit is RES1.
VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VAckCtl</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt.</td>
</tr>
</tbody>
</table>

This bit is an alias of GICV_CTLR.AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of ICC_SRE_EL1.SRE is always 1, this bit is RES0.

VENG1, bit [1]

Virtual Group 1 interrupt enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VENG1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This bit is an alias of ICV_IGRPEN1_EL1.Enable.

VENG0, bit [0]

Virtual Group 0 interrupt enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VENG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This bit is an alias of ICV_IGRPEN0_EL1.Enable.

Accessing the ICH_VMCR_EL2

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings:

MRS <Xt>, ICH_VMCR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x4C8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_VMCR_EL2;
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_VMCR_EL2;
    end
end

MSR ICH_VMCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b110</td>
<td>0b101</td>
<td>0b11</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x4C8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_VMCR_EL2 = X[t];
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_VMCR_EL2 = X[t];
    end

ICH_VTR_EL2, Interrupt Controller VGIC Type Register

The ICH_VTR_EL2 characteristics are:

**Purpose**

Reports supported GIC virtualisation features.

**Configuration**

AArch64 System register ICH_VTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register ICH_VTR[31:0].

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

**Attributes**

ICH_VTR_EL2 is a 64-bit register.

**Field descriptions**

The ICH_VTR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| PRIbits | PREbits | IDbits | SEISA3V | nV4 | TDS | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegs | ListRegions |  

**Bits [63:32]**

Reserved, RES0.

**PRIbits, bits [31:29]**

Priority bits. The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

This field is an alias of ICV_CTLR_EL1.PRIbits.

**PREbits, bits [28:26]**

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR_EL2.PRIbits.

The maximum value of this field is 6, indicating 7 bits of preemption.

This field determines the minimum value of ICH_VMCR_EL2.VBPR0.

**IDbits, bits [25:23]**

The number of virtual interrupt identifier bits supported:
### IDbits

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is an alias of `ICV_CTRLR_EL1.IDbits`.

### SEIS, bit [22]

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The virtual CPU interface logic does not support generation of SEIs.</td>
</tr>
<tr>
<td>1</td>
<td>The virtual CPU interface logic supports generation of SEIs.</td>
</tr>
</tbody>
</table>

This bit is an alias of `ICV_CTRLR_EL1.SEIS`.

### A3V, bit [21]

Affinity 3 Valid. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
<tr>
<td>1</td>
<td>The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
</tbody>
</table>

This bit is an alias of `ICV_CTRLR_EL1.A3V`.

### nV4, bit [20]

Direct injection of virtual interrupts not supported. Possible values are:

<table>
<thead>
<tr>
<th>nV4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The CPU interface logic supports direct injection of virtual interrupts.</td>
</tr>
<tr>
<td>1</td>
<td>The CPU interface logic does not support direct injection of virtual interrupts.</td>
</tr>
</tbody>
</table>

In GICv3 this bit is RES1.

### TDS, bit [19]

Separate trapping of EL1 writes to `ICV_DIR_EL1` supported.

<table>
<thead>
<tr>
<th>TDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Implementation does not support <code>ICH_HCR_EL2.TDIR</code>.</td>
</tr>
<tr>
<td>1</td>
<td>Implementation supports <code>ICH_HCR_EL2.TDIR</code>.</td>
</tr>
</tbody>
</table>

### Bits [18:5]

Reserved, RES0.

### ListRegs, bits [4:0]

The number of implemented List registers, minus one. For example, a value of 0b01111 indicates that the maximum of 16 List registers are implemented.
Accessing the ICH_VTR_EL2

Accesses to this register use the following encodings:

\[ \text{MRS } <\text{Xt}>, \text{ ICH_VTR_EL2} \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else if PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ICH_VTR_EL2;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICH_VTR_EL2;
ICV_AP0R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 0 Registers, n = 0 - 3

The ICV_AP0R<n>_EL1 characteristics are:

**Purpose**

Provides information about virtual Group 0 active priorities.

**Configuration**

AArch64 System register ICV_AP0R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_AP0R<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_AP0R<n>_EL1 is a 64-bit register.

**Field descriptions**

The ICV_AP0R<n>_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | IMPLEMENTATION DEFINED |

**Bits [63:32]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

**Accessing the ICV_AP0R<n>_EL1**

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP0R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP0R2_EL1 and ICV_AP0R3_EL1 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:
ICV\_AP0R\_n\_EL1, Interrupt Controller Virtual Active Priorities Group 0 Registers, n = 0 - 3

- ICV\_AP0R\_n\_EL1.
- ICV\_AP1R\_n\_EL1.

Accesses to this register use the following encodings:

MRS <Xt>, ICV\_AP0R\_n\_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC\_SRE\_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH\_HCR\_EL2.TALL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR\_EL2.FMO == '1' then
    return ICV\_AP0R\_EL1[UInt(op2<1:0>)];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR\_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC\_AP0R\_EL1[UInt(op2<1:0>)];
endif

elsif PSTATE.EL == EL2 then
  if ICC\_SRE\_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR\_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC\_AP0R\_EL1[UInt(op2<1:0>)];
endif

elsif PSTATE.EL == EL3 then
  if ICC\_SRE\_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC\_AP0R\_EL1[UInt(op2<1:0>)];
endif

MSR ICC\_AP0R\_n\_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
  endif
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
  endif
ICV_AP1R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 1 Registers, n = 0 - 3

The ICV_AP1R<n>_EL1 characteristics are:

**Purpose**

Provides information about virtual Group 1 active priorities.

**Configuration**

AArch64 System register ICV_AP1R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_AP1R<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_AP1R<n>_EL1 is a 64-bit register.

**Field descriptions**

The ICV_AP1R<n>_EL1 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:32]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

**Accessing the ICV_AP1R<n>_EL1**

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP1R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP1R2_EL1 and ICV_AP1R3_EL1 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:
ICV_AP1R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 1 Registers, n = 0 - 3

- ICV_AP0R<n>_EL1
- ICV_AP1R<n>_EL1.

Accesses to this register use the following encodings:

MRS <Xt>, ICC_AP1R<n>_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() & !ELUsingAArch32(EL2) & ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif !ELUsingAArch32(EL2) & HCR_EL2.IM0 == '1' then
        return ICV_AP1R_EL1[UInt(op2<1:0>)];
    elseif HaveEL(EL3) & !ELUsingAArch32(EL3) & SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
        end
    else
        return ICC_AP1R_EL1[UInt(op2<1:0>)];
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) & !ELUsingAArch32(EL3) & SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
        end
    else
        return ICC_AP1R_EL1[UInt(op2<1:0>)];
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif SCR_EL3.NS == '0' then
        return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
    else
        return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
    end
MSR ICC_AP1R<n>_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif SCR_EL3.NS == '0' then
        ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
else
    ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0

The ICV_BPR0_EL1 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

**Configuration**

AArch64 System register ICV_BPR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_BPR0[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICV_BPR0_EL1 is a 64-bit register.

**Field descriptions**

The ICV_BPR0_EL1 bit assignments are:

| Bits [63:3] | Reserved, RES0. |
| BinaryPoint, bits [2:0] | The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows: |

<table>
<thead>
<tr>
<th>Binary point value</th>
<th>Group priority field</th>
<th>Subpriority field</th>
<th>Field with binary point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[7:1]</td>
<td>[0]</td>
<td>ggggggg.s</td>
</tr>
<tr>
<td>1</td>
<td>[7:2]</td>
<td>[1:0]</td>
<td>gggggg.sss</td>
</tr>
<tr>
<td>2</td>
<td>[7:3]</td>
<td>[2:0]</td>
<td>gggggg.sss</td>
</tr>
<tr>
<td>3</td>
<td>[7:4]</td>
<td>[3:0]</td>
<td>gggggg.sss</td>
</tr>
<tr>
<td>4</td>
<td>[7:5]</td>
<td>[4:0]</td>
<td>ggg.sssss</td>
</tr>
<tr>
<td>5</td>
<td>[7:6]</td>
<td>[5:0]</td>
<td>gssssss</td>
</tr>
<tr>
<td>6</td>
<td>[7]</td>
<td>[6:0]</td>
<td>sssssss</td>
</tr>
<tr>
<td>7</td>
<td>No preemption</td>
<td>[7:0]</td>
<td>.ssssssss</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ICV_BPR0_EL1**

The minimum binary point value is derived from the number of implemented preemption bits, as shown in the following table:
## Number of implemented preemption bits vs. Minimum value of BPR0

<table>
<thead>
<tr>
<th>Number of implemented preemption bits</th>
<th>Minimum value of BPR0</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

The number of implemented preemption bits is indicated by `ICH_VTR_EL2.PREbits`.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is `UNKNOWN`.

Accesses to this register use the following encodings:

**MRS <Xt>, ICC_BPR0_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

```python
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_BPR0_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_BPR0_EL1;
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_BPR0_EL1;
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_BPR0_EL1;
    endif
end if
```

**MSR ICC_BPR0_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_BPR0_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
    end
The ICV_BPR1_EL1 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

**Configuration**

AArch64 System register ICV_BPR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_BPR1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally unknown values.

**Attributes**

ICV_BPR1_EL1 is a 64-bit register.

**Field descriptions**

The ICV_BPR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignments</th>
<th>RES0</th>
<th>BinaryPoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>63   62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>RES0</td>
<td>BinaryPoint</td>
</tr>
<tr>
<td>31   30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:3]**

Reserved, RES0.

**BinaryPoint, bits [2:0]**

If the GIC is configured to use separate binary point fields for virtual Group 0 and virtual Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

<table>
<thead>
<tr>
<th>Binary point value</th>
<th>Group priority field</th>
<th>Subpriority field</th>
<th>Field with binary point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>1</td>
<td>[7:1]</td>
<td>[0]</td>
<td>gggggggg.s</td>
</tr>
<tr>
<td>2</td>
<td>[7:2]</td>
<td>[1:0]</td>
<td>ggggggg.ss</td>
</tr>
<tr>
<td>3</td>
<td>[7:3]</td>
<td>[2:0]</td>
<td>gggggg.sss</td>
</tr>
<tr>
<td>4</td>
<td>[7:4]</td>
<td>[3:0]</td>
<td>ggggg.sssss</td>
</tr>
<tr>
<td>5</td>
<td>[7:5]</td>
<td>[4:0]</td>
<td>gggg.sssss</td>
</tr>
<tr>
<td>6</td>
<td>[7:6]</td>
<td>[5:0]</td>
<td>gg.sssssss</td>
</tr>
<tr>
<td>7</td>
<td>[7]</td>
<td>[6:0]</td>
<td>g.sssssssss</td>
</tr>
</tbody>
</table>

Writing 0 to this field will set this field to its reset value.

If ICV_CTLR_EL1.CBPR is set to 1, Non-secure EL1 reads return ICV_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL1 writes are ignored.
If `ICV_CTLR_EL1.CBPR` is set to 1, Secure EL1 reads return `ICV_BPR0_EL1`. Secure EL1 writes modify `ICV_BPR0_EL1`.

This field resets to an IMPLEMENTATION DEFINED non-zero value.

**Accessing the ICV_BPR1_EL1**

The reset value is IMPLEMENTATION DEFINED, but is equal to the minimum value of `ICV_BPR0_EL1` plus one.

An attempt to program the binary point field to a value less than the reset value sets the field to the reset value.

Accesses to this register use the following encodings:

```plaintext
MRS <Xt>, ICC_BPR1_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
```

If `PSTATE_EL == EL0` then UNDEFINED;
elsif `PSTATE_EL == EL1` then
  if `ICC_SRE_EL1.SRE == '0'` then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif `EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1'` then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif `EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1'` then
    return ICV_BPR1_EL1;
  elsif `HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1'` then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif `HaveEL(EL3)` then
    if `SCR_EL3.NS == '0'` then
      return ICC_BPR1_EL1_S;
    else
      return ICC_BPR1_EL1_NS;
    end if
  else
    return ICC_BPR1_EL1;
  end if
elsif `PSTATE_EL == EL2` then
  if `ICC_SRE_EL2.SRE == '0'` then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif `HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1'` then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif `HaveEL(EL3)` then
    if `SCR_EL3.NS == '0'` then
      return ICC_BPR1_EL1_S;
    else
      return ICC_BPR1_EL1_NS;
    end if
  else
    return ICC_BPR1_EL1;
  end if
elsif `PSTATE_EL == EL3` then
  if `ICC_SRE_EL3.SRE == '0'` then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif `HaveEL(EL3)` then
    if `SCR_EL3.NS == '0'` then
      return ICC_BPR1_EL1_S;
    else
      return ICC_BPR1_EL1_NS;
    end if
  else
    return ICC_BPR1_EL1;
  end if
```

MSR ICC_BPR1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if SCR_EL1 NS == '0' then
    ICC_BPR1_EL1_S = X[t];
  else
    ICC_BPR1_EL1_NS = X[t];
else
  ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if SCR_EL2 NS == '0' then
    ICC_BPR1_EL1_S = X[t];
  else
    ICC_BPR1_EL1_NS = X[t];
else
  ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  if SCR_EL3 NS == '0' then
    ICC_BPR1_EL1_S = X[t];
  else
    ICC_BPR1_EL1_NS = X[t];
ICV_CTLR_EL1, Interrupt Controller Virtual Control Register

The ICV_CTLR_EL1 characteristics are:

**Purpose**

Controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

**Configuration**

AArch64 System register ICV_CTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_CTLR[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICV_CTLR_EL1 is a 64-bit register.

**Field descriptions**

The ICV_CTLR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-20</td>
<td><strong>Reserved, RES0</strong></td>
</tr>
<tr>
<td>19</td>
<td><strong>ExtRange</strong></td>
</tr>
<tr>
<td>18-15</td>
<td><strong>PRIbits</strong></td>
</tr>
<tr>
<td>14-11</td>
<td><strong>IDbits</strong></td>
</tr>
<tr>
<td>10-9</td>
<td><strong>A3V</strong></td>
</tr>
<tr>
<td>8-5</td>
<td><strong>RSS</strong></td>
</tr>
<tr>
<td>4-0</td>
<td><strong>ExtRange</strong></td>
</tr>
</tbody>
</table>

**Bits [63:20]**

Reserved, RES0.

**ExtRange, bit [19]**

Extended INTID range (read-only).

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0      | CPU interface does not support INTIDs in the range 1024..8191.  
  - Behaviour is **UNPREDICTABLE** if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.  
  **Note**  
  Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them. |
| 0b1      | CPU interface supports INTIDs in the range 1024..8191  
  - All INTIDs in the range 1024..8191 are treated as requiring deactivation. |

ICV_CTLR_EL1.ExtRange is an alias of ICC_CTLR_EL1.ExtRange.
RSS, bit [18]

Range Selector Support. Possible values are:

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 15 are supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 255 are supported.</td>
</tr>
</tbody>
</table>

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
</tbody>
</table>

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the virtual CPU interface supports local generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic does not support local generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports local generation of SEIs.</td>
</tr>
</tbody>
</table>

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of virtual interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

PRibits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation must implement at least 32 levels of physical priority (5 priority bits).

Note

This field always returns the number of priority bits implemented.

The division between group priority and subpriority is defined in the binary point registers ICV_BPR0_EL1 and ICV_BPR1_EL1.

Bits [7:2]

Reserved, RES0.
EOI mode, bit [1]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

<table>
<thead>
<tr>
<th>EOI mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts:

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 0b111. Writes to ICV_BPR1_EL1 are ignored.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Accessing the ICV_CTLR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ICC_CTLR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TC == '1' then
        return ICC_CTR_EL1;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_CTLR_EL1;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_CTLR_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
        end if
    end if;
else
    return ICC_CTLR_EL1;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
        end if
    end if;
else
    return ICC_CTLR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
        end if
    end if;
else
    return ICC_CTLR_EL1;
end if;

MSR ICC_CTLR_EL1, <Xt>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        ICH_HCR_EL2.TC = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        HCR_EL2.IMO = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        HCR_EL2.FMO = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1.S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
        endif
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1.S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
        endif
    else
        ICC_CTLR_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif SCR_EL3.NS == '0' then
        ICC_CTLR_EL1.S = X[t];
    else
        ICC_CTLR_EL1_NS = X[t];
    endif
else
    ICC_CTLR_EL1 = X[t];
endif
ICV_DIR_EL1, Interrupt Controller Deactivate Virtual Interrupt Register

The ICV_DIR_EL1 characteristics are:

**Purpose**

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified virtual interrupt.

**Configuration**

AArch64 System register ICV_DIR_EL1 bits [31:0] performs the same function as AArch32 System register ICV_DIR[31:0].

**Attributes**

ICV_DIR_EL1 is a 64-bit register.

**Field descriptions**

The ICV_DIR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the virtual interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_DIR_EL1**

When EOImode == 0, writes are ignored. In systems supporting system error generation, an implementation might generate an SEI.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TDIR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_DIR_EL1 = X[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_DIR_EL1 = X[t];
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];
    endif
elseif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];
    endif
elseif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];
    endif
The ICV_EOIR0_EL1 characteristics are:

**Purpose**
A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 0 interrupt.

**Configuration**
AArch64 System register ICV_EOIR0_EL1 performs the same function as AArch32 System register ICV_EOIR0.

**Attributes**
ICV_EOIR0_EL1 is a 64-bit register.

**Field descriptions**
The ICV_EOIR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23:0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]
The INTID from the corresponding ICV_IAR0_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR.EOiMode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR.EOiMode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR_EL1 to deactivate the virtual interrupt.

**Accessing the ICV_EOIR0_EL1**
A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR0_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings:

```
MSR ICC_EOIR0_EL1, <Xt>
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_EOIR0_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t];
    endif
ICV_EOIR1_EL1, Interrupt Controller Virtual End Of Interrupt Register 1

The ICV_EOIR1_EL1 characteristics are:

**Purpose**

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 1 interrupt.

**Configuration**

AArch64 System register ICV_EOIR1_EL1 performs the same function as AArch32 System register ICV_EOIR1.

**Attributes**

ICV_EOIR1_EL1 is a 64-bit register.

**Field descriptions**

The ICV_EOIR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>INTID, bits [23:0]</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID from the corresponding ICV_IAR1_EL1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR_EL1 to deactivate the virtual interrupt.

**Accessing the ICV_EOIR1_EL1**

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR1_EL1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings:

```
MSR ICC_EOIR1_EL1, <Xt>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>

Page 887
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_EOIR1_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
}
ICV_HPPIR0_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

The ICV_HPPIR0_EL1 characteristics are:

**Purpose**

Indicates the highest priority pending virtual Group 0 interrupt on the virtual CPU interface.

**Configuration**

AArch64 System register ICV_HPPIR0_EL1 performs the same function as AArch32 System register ICV_HPPIR0.

**Attributes**

ICV_HPPIR0_EL1 is a 64-bit register.

**Field descriptions**

The ICV_HPPIR0_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_HPPIR0_EL1**

Accesses to this register use the following encodings:

\[
\text{MRS} \ <Xt>, \ \text{ICC\_HPPIR0\_EL1}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_HPPIR0_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_HPPIR0_EL1;
    elsif PSTATE.EL == EL2 then
        if ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR0_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR0_EL1;
ICV_HPPIR1_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

The ICV_HPPIR1_EL1 characteristics are:

**Purpose**

Indicates the highest priority pending virtual Group 1 interrupt on the virtual CPU interface.

**Configuration**

AArch64 System register ICV_HPPIR1_EL1 performs the same function as AArch32 System register ICV_HPPIR1.

**Attributes**

ICV_HPPIR1_EL1 is a 64-bit register.

**Field descriptions**

The ICV_HPPIR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>RES0</td>
<td>INTID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_HPPIR1_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ICC_HPPIR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IM0 == '1' then
        return ICV_HPPIR1_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL2 then
        if ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;
    else
        return ICC_HPPIR1_EL1;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ICV_IAR0_EL1, Interrupt Controller Virtual Interrupt Acknowledge Register 0

The ICV_IAR0_EL1 characteristics are:

**Purpose**

The PE reads this register to obtain the INTID of the signaled virtual Group 0 interrupt. This read acts as an acknowledge for the interrupt.

**Configuration**

AArch64 System register ICV_IAR0_EL1 performs the same function as AArch32 System register ICV_IAR0.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

**Attributes**

ICV_IAR0_EL1 is a 64-bit register.

**Field descriptions**

The ICV_IAR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit 63</th>
<th>Bit 62</th>
<th>Bit 61</th>
<th>Bit 60</th>
<th>Bit 59</th>
<th>Bit 58</th>
<th>Bit 57</th>
<th>Bit 56</th>
<th>Bit 55</th>
<th>Bit 54</th>
<th>Bit 53</th>
<th>Bit 52</th>
<th>Bit 51</th>
<th>Bit 50</th>
<th>Bit 49</th>
<th>Bit 48</th>
<th>Bit 47</th>
<th>Bit 46</th>
<th>Bit 45</th>
<th>Bit 44</th>
<th>Bit 43</th>
<th>Bit 42</th>
<th>Bit 41</th>
<th>Bit 40</th>
<th>Bit 39</th>
<th>Bit 38</th>
<th>Bit 37</th>
<th>Bit 36</th>
<th>Bit 35</th>
<th>Bit 34</th>
<th>Bit 33</th>
<th>Bit 32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_IAR0_EL1**

Accesses to this register use the following encodings:
MRS <Xt>, ICC_IAR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICC_IAR0_EL1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR0_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR0_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR0_EL1;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ICV_IAR1_EL1, Interrupt Controller Virtual Interrupt Acknowledge Register 1

The ICV_IAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICV_IAR1_EL1 performs the same function as AArch32 System register ICV_IAR1.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

Attributes

ICV_IAR1_EL1 is a 64-bit register.

Field descriptions

The ICV_IAR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR_EL1.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing the ICV_IAR1_EL1

Accesses to this register use the following encodings:
MRS <Xt>, ICC_IAR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_IAR1_EL1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR1_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR1_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return ICC_IAR1_EL1;
ICV_IGRPEN0_EL1, Interrupt Controller Virtual Interrupt Group 0 Enable register

The ICV_IGRPEN0_EL1 characteristics are:

**Purpose**

Controls whether virtual Group 0 interrupts are enabled or not.

**Configuration**

AArch64 System register ICV_IGRPEN0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_IGRPEN0[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_IGRPEN0_EL1 is a 64-bit register.

**Field descriptions**

The ICV_IGRPEN0_EL1 bit assignments are:

| Bits [63:1] | Reserved, RES0. |
| Bits [0] | Enables virtual Group 0 interrupts. |

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the ICV_IGRPEN0_EL1**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS &lt;Xt&gt;, ICC_IGRPEN0_EL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
</tr>
<tr>
<td>0b11</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_IGRPEN0_EL1;
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN0_EL1;
    endif
elseif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN0_EL1;
    endif
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t];
    endif
    MSR ICC_IGRPEN0_EL1, <Xt>

    | op0 | op1 | CRn | CRm | op2 |
    |-----|-----|-----|-----|-----|
    | 0b11 | 0b000 | 0b1100 | 0b1100 | 0b110 |

    if PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if ICC_SRE_EL1.SRE == '0' then
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
            HFGWR_EL2.ICC_IGRPENn_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.FMO == '1' then
            ICV_IGRPEN0_EL1 = X[t];
        elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.FIQ == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];
        endif
    elsif PSTATE.EL == EL2 then
        if ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.FIQ == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];
        endif
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];
        endif
    else
        ICC_IGRPEN0_EL1 = X[t];
    endif
ICV_IGRPEN1_EL1, Interrupt Controller Virtual Interrupt Group 1 Enable register

The ICV_IGRPEN1_EL1 characteristics are:

**Purpose**

Controls whether virtual Group 1 interrupts are enabled for the current Security state.

**Configuration**

AArch64 System register ICV_IGRPEN1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_IGRPEN1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_IGRPEN1_EL1 is a 64-bit register.

**Field descriptions**

The ICV_IGRPEN1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | Enable |

**Bits [63:1]**

Reserved, RES0.

**Enable, bit [0]**

Enables virtual Group 1 interrupts.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the ICV_IGRPEN1_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ICC_IGRPEN1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.ICC_IGRPENn_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IGRPEN1_EL1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
    else
        return ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
    else
        return ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
end

MSR ICC_IGRPEN1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.ICC_IGRPENn_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICC_IGRPEN1_EL1 = X[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      ICC_IGRPEN1_EL1_S = X[t];
    else
      ICC_IGRPEN1_EL1_NS = X[t];
    end
  else
    ICC_IGRPEN1_EL1 = X[t];
  end
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      ICC_IGRPEN1_EL1_S = X[t];
    else
      ICC_IGRPEN1_EL1_NS = X[t];
    end
  else
    ICC_IGRPEN1_EL1 = X[t];
  end
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    if SCR_EL3.NS == '0' then
      ICC_IGRPEN1_EL1_S = X[t];
    else
      ICC_IGRPEN1_EL1_NS = X[t];
    end
  end
ICV_PMR_EL1, Interrupt Controller Virtual Interrupt Priority Mask Register

The ICV_PMR_EL1 characteristics are:

**Purpose**

Provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

**Configuration**

AArch64 System register ICV_PMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register ICV_PMR[31:0].

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. See Observability of the effects of accesses to the GIC registers, for more information.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_PMR_EL1 is a 64-bit register.

**Field descriptions**

The ICV_PMR_EL1 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 |

**Bits [63:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The priority mask level for the virtual CPU interface. If the priority of a virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

The possible priority field values are as follows:

<table>
<thead>
<tr>
<th>Implemented priority bits</th>
<th>Possible priority field values</th>
<th>Number of priority levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:0]</td>
<td>0x00-0xFF (0-255), all values</td>
<td>256</td>
</tr>
<tr>
<td>[7:1]</td>
<td>0x00-0xFE (0-254), even values only</td>
<td>128</td>
</tr>
<tr>
<td>[7:2]</td>
<td>0x00-0xFC (0-252), in steps of 4</td>
<td>64</td>
</tr>
<tr>
<td>[7:3]</td>
<td>0x00-0xF8 (0-248), in steps of 8</td>
<td>32</td>
</tr>
<tr>
<td>[7:4]</td>
<td>0x00-0xF0 (0-240), in steps of 16</td>
<td>16</td>
</tr>
</tbody>
</table>
Unimplemented priority bits are RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the ICV_PMR_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, ICC_PMR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elses PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
elses if EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elses if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_PMR_EL1;
elses if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_PMR_EL1;
elses if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return ICC_PMR_EL1;
elses PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elses if EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return ICC_PMR_EL1;
elses PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return ICC_PMR_EL1;
```

**MSR ICC_PMR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICC_PMR_EL1 = X[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICC_PMR_EL1 = X[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t];
    end
**ICV_RPR_EL1, Interrupt Controller Virtual Running Priority Register**

The ICV_RPR_EL1 characteristics are:

**Purpose**

Indicates the Running priority of the virtual CPU interface.

**Configuration**

AArch64 System register ICV_RPR_EL1 performs the same function as AArch32 System register ICV_RPR.

**Attributes**

ICV_RPR_EL1 is a 64-bit register.

**Field descriptions**

The ICV_RPR_EL1 bit assignments are:

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
</tr>
<tr>
<td>48</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active virtual interrupt.

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

**Note**

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

**Accessing the ICV_RPR_EL1**

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings:
MRS <Xt>, ICC_RPR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ICC_SRE_EL1.SRE == '0' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICC_RPR_EL1;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICC_RPR_EL1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return ICC_RPR_EL1;
elsif PSTATE.EL == EL2 then
  if ICC_SRE_EL2.SRE == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return ICC_RPR_EL1;
elsif PSTATE.EL == EL3 then
  if ICC_SRE_EL3.SRE == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return ICC_RPR_EL1;
ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0

The ID_AA64AFR0_EL1 characteristics are:

**Purpose**

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64AFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64AFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [31:28]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [27:24]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [23:20]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [19:16]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [15:12]**

IMPLEMENTATION DEFINED.
IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Accessing the ID_AA64AFR0_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_AA64AFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return ID_AA64AFR0_EL1;
  elsif PSTATE.EL == EL2 then
    return ID_AA64AFR0_EL1;
  elsif PSTATE.EL == EL3 then
    return ID_AA64AFR0_EL1;
ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1

The ID_AA64AFR1_EL1 characteristics are:

**Purpose**

Reserved for future expansion of information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64AFR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64AFR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Positions</th>
<th>Bit Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>Reserved RES0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Reserved, RES0.

**Accessing the ID_AA64AFR1_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64AFR1_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64AFR1_EL1;
elsif PSTATE.EL == EL2 then
    return ID_AA64AFR1_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64AFR1_EL1;
The ID_AA64DFR0_EL1 characteristics are:

**Purpose**

Provides top level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

The external register `EDDFR` gives information from this register.

**Attributes**

ID_AA64DFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64DFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:52]</td>
<td>Reserved, <code>RES0</code></td>
</tr>
<tr>
<td>[51:48]</td>
<td><code>MTPMU</code></td>
</tr>
<tr>
<td>[47:44]</td>
<td><code>TraceBuffer</code></td>
</tr>
<tr>
<td>[43:40]</td>
<td><code>TraceFilt</code></td>
</tr>
<tr>
<td>[39:36]</td>
<td><code>DoubleLock</code></td>
</tr>
<tr>
<td>[35:32]</td>
<td><code>PMSVer</code></td>
</tr>
<tr>
<td>[31:24]</td>
<td><code>CTX_CMPs</code></td>
</tr>
<tr>
<td>[23:20]</td>
<td>Reserved, <code>RES0</code></td>
</tr>
<tr>
<td>[19:16]</td>
<td><code>WRPs</code></td>
</tr>
<tr>
<td>[15:12]</td>
<td>Reserved, <code>RES0</code></td>
</tr>
<tr>
<td>[11:8]</td>
<td><code>BRPs</code></td>
</tr>
<tr>
<td>[7:4]</td>
<td><code>PMUVer</code></td>
</tr>
<tr>
<td>[3:0]</td>
<td><code>TraceVer</code></td>
</tr>
</tbody>
</table>

**Bits [63:52]**

Reserved, `RES0`.

**MTPMU, bits [51:48]**

From Armv8.6:

Multi-threaded PMU extension. Defined values are:

<table>
<thead>
<tr>
<th>MTPMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ARMv8.6-MTPMU not implemented. If PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER&lt;n&gt;_EL0.MT are read/write or <code>RES0</code>.</td>
</tr>
<tr>
<td>0b0001</td>
<td>ARMv8.6-MTPMU implemented and PMEVTYPER&lt;n&gt;_EL0.MT are read/write. When ARMv8.6-MTPMU is disabled, the Effective values of PMEVTYPER&lt;n&gt;_EL0.MT are 0.</td>
</tr>
<tr>
<td>0b1111</td>
<td>ARMv8.6-MTPMU not implemented. If PMUv3 is implemented, PMEVTYPER&lt;n&gt;_EL0.MT are <code>RES0</code>.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-MTPMU implements the functionality identified by the value 0b0001.

In an Armv8.6-compliant implementation that includes PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include PMUv3, the value 0b0001 is not permitted.
Otherwise:

Reserved, RES0.

**TraceBuffer, bits [47:44]**

When TRBE is implemented:

Trace Buffer Extension version. Defined values are:

<table>
<thead>
<tr>
<th>TraceBuffer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Trace Buffer Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Trace Buffer Extension implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**TraceFilt, bits [43:40]**

From Armv8.4:

Armv8.4 Self-hosted Trace Extension version. Defined values are:

<table>
<thead>
<tr>
<th>TraceFilt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Armv8.4 Self-hosted Trace Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Armv8.4 Self-hosted Trace Extension implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-Trace implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

**DoubleLock, bits [39:36]**

OS Double Lock implemented. Defined values are:

<table>
<thead>
<tr>
<th>DoubleLock</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>OS Double Lock implemented. OSDLR_EL1 is RW.</td>
</tr>
<tr>
<td>0b1111</td>
<td>OS Double Lock not implemented. OSDLR_EL1 is RAZ/WI.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.0-DoubleLock implements the functionality identified by the value 0b0000.

**PMSVer, bits [35:32]**

From Armv8.2:

Statistical Profiling Extension version. Defined values are:
<table>
<thead>
<tr>
<th>PMSVer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Statistical Profiling Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Statistical Profiling Extension implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001 and also includes support for:</td>
</tr>
<tr>
<td></td>
<td>• The Event packet Alignment flag.</td>
</tr>
<tr>
<td></td>
<td>• If SVE is implemented, the Scalable Vector extensions to Statistical Profiling.</td>
</tr>
</tbody>
</table>

All other values are reserved.

SPE implements the functionality identified by the value 0b0001.

ARMv8.3-SPE implements the functionality added by the value 0b0010. If ARMv8.3-SPE is implemented, then ID_AA64FR0_EL1.PMSVer is not permitted to read as 0b0000 or 0b0001.

Otherwise:

Reserved, RES0.

**CTX_CMPs, bits [31:28]**

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

**Bits [27:24]**

Reserved, RES0.

**WRPs, bits [23:20]**

Number of watchpoints, minus 1. The value of 0b0000 is reserved.

**Bits [19:16]**

Reserved, RES0.

**BRPs, bits [15:12]**

Number of breakpoints, minus 1. The value of 0b0000 is reserved.

**PMUVer, bits [11:8]**

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the Alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Defined values are:
### PMUVer

<table>
<thead>
<tr>
<th>PMUVer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Performance Monitors Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Performance Monitors Extension implemented, PMUv3.</td>
</tr>
</tbody>
</table>
| 0b0100 | PMUv3 for Armv8.1. As 0b0001, and also includes support for:  
  - Extended 16-bit PMEVTYPER\textsubscript{<n>}_EL0 evtCount field.  
  - If EL2 is implemented, the MDCR\_EL2 HPMD control bit. |
| 0b0101 | PMUv3 for Armv8.2. As 0b0100 and also includes support for the PMMIR\_EL1 register. |
| 0b0110 | PMUv3 for Armv8.4. As 0b0101 and also includes support for:  
  - 64-bit event counters.  
  - If EL2 is implemented, the MDCR\_EL2 HCCD control bit.  
  - If EL3 is implemented, the MDCR\_EL3 SCCD control bit. |
| 0b1111 | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value in new implementations. |

All other values are reserved.

ARMv8.1-PMU implements the functionality identified by the value 0b0100.

ARMv8.4-PMU implements the functionality identified by the value 0b0101.

ARMv8.5-PMU implements the functionality identified by the value 0b0110.

In an Armv8.1-compliant implementation that includes PMUv3, the value 0b0001 is not permitted.

In an Armv8.4-compliant implementation that includes PMUv3, the value 0b0100 is not permitted.

In an Armv8.5-compliant implementation that includes PMUv3, the value 0b0101 is not permitted.

### TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a PE trace unit is implemented. Defined values are:

<table>
<thead>
<tr>
<th>TraceVer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PE trace unit System registers not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PE trace unit System registers implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When PE trace unit System registers are implemented, see TRCIDR1 for tracing capabilities of the trace unit.

### DebugVer, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture. Defined values are:

<table>
<thead>
<tr>
<th>DebugVer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0110</td>
<td>Armv8 debug architecture.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv8 debug architecture with Virtualization Host Extensions.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Armv8.2 debug architecture</td>
</tr>
<tr>
<td>0b1001</td>
<td>Armv8.4 debug architecture</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-Debug adds the functionality identified by the value 0b1000.

- If Armv8.1-VHE is not implemented the only permitted value is 0b0110.
- In an Armv8.0 implementation the value 0b0100 is not permitted.

### Accessing the ID_AA64DFR0\_EL1

Accesses to this register use the following encodings:
MRS <Xt>, ID_AA64DFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end
else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL2 then
    return ID_AA64DFR0_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64DFR0_EL1;
The ID_AA64DFR1_EL1 characteristics are:

**Purpose**

Reserved for future expansion of top level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64DFR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64DFR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 |

Bits [63:0]

Reserved, RES0.

**Accessing the ID_AA64DFR1_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64DFR1_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64DFR1_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_AA64DFR1_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_AA64DFR1_EL1;
```
The ID_AA64ISAR0_EL1 characteristics are:

**Purpose**

Provides information about the instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64ISAR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64ISAR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:60 RNDR</td>
<td>Indicates support for Random Number instructions in AArch64 state. Defined values are:</td>
</tr>
<tr>
<td>60:56 TLB</td>
<td>From Armv8.5:</td>
</tr>
<tr>
<td>60:2 RDM</td>
<td>All other values are reserved.</td>
</tr>
<tr>
<td>51:0 Atomic</td>
<td>ARMv8.5-RNG implements the functionality identified by the value 0b0001.</td>
</tr>
<tr>
<td>55:40 FHM</td>
<td>From Armv8.5, the permitted values are 0b0000 and 0b0001.</td>
</tr>
<tr>
<td>49:4 RES0</td>
<td>Otherwise:</td>
</tr>
<tr>
<td>48:30 TME</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46:24 CRC32</td>
<td></td>
</tr>
<tr>
<td>44:8 SHA2</td>
<td></td>
</tr>
<tr>
<td>42:4 SHA1</td>
<td></td>
</tr>
<tr>
<td>40:2 AES</td>
<td></td>
</tr>
<tr>
<td>38:0 RES0</td>
<td></td>
</tr>
</tbody>
</table>

**RNDR, bits [63:60]**

From Armv8.5:

Indicates support for Random Number instructions in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>RNDR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Random Number instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>RNDR and RNDRRS registers are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.5-RNG implements the functionality identified by the value 0b0001.

From Armv8.5, the permitted values are 0b0000 and 0b0001.

**TLB, bits [59:56]**

From Armv8.4:

Indicates support for Outer shareable and TLB range maintenance instructions. Defined values are:
### TLB

<table>
<thead>
<tr>
<th>TLB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Outer shareable and TLB range maintenance instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Outer shareable TLB maintenance instructions are implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Outer shareable and TLB range maintenance instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-TLBI implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.4, the only permitted value is 0b0010.

**Otherwise:**

Reserved, RES0.

### TS, bits [55:52]

Indicates support for flag manipulation instructions. Defined values are:

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No flag manipulation instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CFINV, RMIF, SETF16, and SETF8 instructions are implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>CFINV, RMIF, SETF16, SETF8, AXFLAG, and XAFLAG instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-CondM implements the functionality identified by the value 0b0001.

ARMv8.5-CondM implements the functionality identified by the value 0b0010.

From Armv8.4, the only permitted value is 0b0001.

From Armv8.5, the only permitted value is 0b0010.

### FHM, bits [51:48]

**From Armv8.2:**

Indicates whether FMLAL and FMLSL instructions are implemented.

<table>
<thead>
<tr>
<th>FHM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>FMLAL and FMLSL instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FMLAL and FMLSL instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-FHM implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

**Otherwise:**

Reserved, RES0.

### DP, bits [47:44]

**From Armv8.2:**

Dot Product instructions implemented in AArch64 state. Defined values are:
<table>
<thead>
<tr>
<th>DP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Dot Product instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>UDOT and SDOT instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-DotProd implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

Otherwise:

Reserved, RES0.

**SM4, bits [43:40]**

From Armv8.2:

SM4 instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SM4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SM4 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SM4E and SM4EKEY instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If ARMv8.2-SM is not implemented the value 0b0001 is reserved.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM4.

Otherwise:

Reserved, RES0.

**SM3, bits [39:36]**

From Armv8.2:

SM3 instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SM3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SM3 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B,</td>
</tr>
<tr>
<td></td>
<td>SM3PARTW1, and SM3PARTW2 instructions ...</td>
</tr>
</tbody>
</table>

All other values are reserved.

If ARMv8.2-SM is not implemented the value 0b0001 is reserved.

ARMv8.2-SM implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM3.

Otherwise:

Reserved, RES0.
SHA3, bits [35:32]

From Armv8.2:

SHA3 instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SHA3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA3 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EOR3, RAX1, XAR, and BCAX instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If Armv8.2-SHA is not implemented the value 0b0001 is reserved.

ARMv8.2-SHA implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, then this field must have the value 0b0000.

If the value of this field is 0b0001, then ID_AA64ISAR0_EL1.SHA2 must have the value 0b0010.

Otherwise:

Reserved, RES0.

RDM, bits [31:28]

From Armv8.1:

SQRDMLAH and SQRDMLSH instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>RDM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SQRDMLAH and SQRDMLSH instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SQRDMLAH and SQRDMLSH instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-RDMA implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

TME, bits [27:24]

When TME is implemented:

Indicates whether TME instructions are implemented:

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>TME instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>TCANCEL, TCOMMIT, TSTART, and TTEST instructions are implemented.</td>
</tr>
</tbody>
</table>

If HCR_EL2.TME == 0, reads of this field at EL1 return 0.

If SCR_EL3.TME == 0, reads of this field at EL1 or EL2 return 0.

All other values are reserved.
Otherwise:
Reserved, RES0.

Atomic, bits [23:20]

From Armv8.1:

Atomic instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>Atomic</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Atomic instructions implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>LDADD, LDCLR, LDEOR, LDSET, LDSMAX, LDSMIN, LDUMAX, LDUMIN, CAS, CASP, and SWP instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-LSE implements the functionality identified by the value 0b0010.

From Armv8.1, the only permitted value is 0b0010.

Otherwise:
Reserved, RES0.

CRC32, bits [19:16]

CRC32 instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>CRC32</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No CRC32 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CRC32B, CRC32H, CRC32W, CRC32X, CRC32CB, CRC32CH, CRC32CW, and CRC32CX instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.1, the only permitted value is 0b0001.

SHA2, bits [15:12]

SHA2 instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SHA2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA2 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 instructions implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, plus SHA512H, SHA512H2, SHA512SU0, and SHA512SU1 instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If ARMv8.2-SHA is not implemented the value 0b0010 is reserved.

From Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, then this field must have the value 0b0000.

If the value of this field is 0b0010, then ID_AA64ISAR0_EL1.SHA3 must have the value 0b0001.
SHA1, bits [11:8]

SHA1 instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SHA1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA1 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA2 is 0b0000, then this field must have the value 0b0000.

AES, bits [7:4]

AES instructions implemented in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>AES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No AES instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AESE, AESD, AESMC, and AESIMC instructions implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, plus PMULL/PMULL2 instructions operating on 64-bit data quantities.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8, the permitted values are 0b0000, 0b0001, and 0b0010.

Bits [3:0]

Reserved, RES0.

Accessing the ID_AA64ISAR0_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_AA64ISAR0_EL1

```
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_AA64ISAR0_EL1;
elsif PSTATE.EL == EL2 then
  return ID_AA64ISAR0_EL1;
elsif PSTATE.EL == EL3 then
  return ID_AA64ISAR0_EL1;
```

Page 924
ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1

The ID_AA64ISAR1_EL1 characteristics are:

**Purpose**

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64ISAR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64ISAR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55-52</td>
<td>I8MM, indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions in AArch64 state. Defined values of this field are:</td>
</tr>
<tr>
<td>51-48</td>
<td>BIMM</td>
</tr>
<tr>
<td>47-44</td>
<td>SPECRES</td>
</tr>
<tr>
<td>43-40</td>
<td>SB</td>
</tr>
<tr>
<td>39-36</td>
<td>APINTTS</td>
</tr>
<tr>
<td>35-32</td>
<td>FRINTTS</td>
</tr>
<tr>
<td>31-28</td>
<td>GPI</td>
</tr>
<tr>
<td>27-24</td>
<td>GPA</td>
</tr>
<tr>
<td>23-20</td>
<td>LRCPC</td>
</tr>
<tr>
<td>19-16</td>
<td>FCMA</td>
</tr>
<tr>
<td>15-12</td>
<td>JSCVT</td>
</tr>
<tr>
<td>11-8</td>
<td>API</td>
</tr>
<tr>
<td>7-4</td>
<td>APA</td>
</tr>
<tr>
<td>3-0</td>
<td>DPB</td>
</tr>
</tbody>
</table>

**Bits [63:56]**

- Reserved, RES0.

**I8MM, bits [55:52]**

For Armv8.2:

Indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions in AArch64 state. Defined values of this field are:

<table>
<thead>
<tr>
<th>I8MM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Int8 matrix multiplication instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SMMLA, SUDOT, UMMLA, USMMLA, and USDOT instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-I8MM implements the functionality identified by 0b0001.

From Armv8.6, the only permitted value is 0b0001.

**Otherwise:**

Reserved, RES0.
DGH, bits [51:48]

From Armv8.6:

Indicates presence of the Data Gathering Hint instruction. Defined values are:

<table>
<thead>
<tr>
<th>DGH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Data Gathering Hint is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Data Gathering Hint is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.0-DGH implements the functionality identified by 0b0001.

From ARMv8.0, the permitted values are 0b0000 and 0b0001.

If the DGH instruction has no effect in preventing the merging of memory accesses, the value of this field is 0.

Otherwise:

Reserved, RES0.

BF16, bits [47:44]

From Armv8.2:

Indicates support for Advanced SIMD and floating-point BFloat16 instructions in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>BF16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>BFloat16 instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>BFDOT, BFMLAL, BFMLAL2, BFMMMLA, BFCVT, and BFCVT2 instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-BF16 implements the functionality identified by 0b0001.

From ARMv8.6, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

SPECRES, bits [43:40]

Speculation invalidation instruction support in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SPECRES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>CFP RCTX, DVP RCTX, and CPP RCTX instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CFP RCTX, DVP RCTX, and CPP RCTX instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [39:36]

SB instruction support in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SB instruction is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SB instruction is implemented.</td>
</tr>
</tbody>
</table>
All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

**FRINTTS, bits [35:32]**

**From Armv8.5:**

Indicates whether FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented. Defined values are:

<table>
<thead>
<tr>
<th>FRINTTS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

**Otherwise:**

Reserved, RES0.

**GPI, bits [31:28]**

**From Armv8.3:**

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for generic code authentication, in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>GPI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Generic Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Generic Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This involves the PACGA instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPA is non-zero, this field must have the value 0b0000.

**Otherwise:**

Reserved, RES0.

**GPA, bits [27:24]**

**From Armv8.3:**

Indicates whether QARMA or Architected algorithm is implemented in the PE for generic code authentication, in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>GPA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Generic Authentication using an Architected algorithm is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Generic Authentication using the QARMA algorithm is implemented. This involves the PACGA instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001.
If the value of ID_AA64ISAR1_EL1.GPI is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.

**LRPC, bits [23:20]**

From Armv8.4:

Indicates support for weaker release consistency, RCpc based model. Defined values are:

<table>
<thead>
<tr>
<th>LRPC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The LDAPUR*, STLUR*, and LDAPR* instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The LDAPR* instructions are implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The LDAPUR*, STLUR*, and LDAPR* instructions are implemented.</td>
</tr>
</tbody>
</table>

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

In Armv8.3, the only permitted value is 0b0001. ARMv8.3-RCPC implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0010. ARMv8.4-RCPC implements the functionality identified by the value 0b0010.

All other values are reserved.

From Armv8.3:

Indicates support for weaker release consistency, RCpc based model. Defined values are:

<table>
<thead>
<tr>
<th>LRPC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The LDAPRB, LDAPRH and LDAPR instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The LDAPRB, LDAPRH and LDAPR instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.3-RCPC implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

In Armv8.3, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

**FCMA, bits [19:16]**

From Armv8.3:

Indicates support for complex number addition and multiplication, where numbers are stored in vectors. Defined values are:

<table>
<thead>
<tr>
<th>FCMA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The FCMLA and FCADD instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The FCMLA and FCADD instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.
ARMv8.3-CompNum implements the functionality identified by the value $0b0001$.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is $0b0000$.

From Armv8.3, the only permitted value is $0b0001$.

Otherwise:

Reserved, RES0.

**JSCVT, bits [15:12]**

*From Armv8.3:*

Indicates support for javascript conversion from double precision floating point values to integers in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>JSCVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0b0000$</td>
<td>The FJCVTZS instruction is not implemented.</td>
</tr>
<tr>
<td>$0b0001$</td>
<td>The FJCVTZS instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.3.JSConv implements the functionality identified by $0b0001$.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is $0b0000$.

From Armv8.3, the only permitted value is $0b0001$.

Otherwise:

Reserved, RES0.

**API, bits [11:8]**

*From Armv8.3:*

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the PACGA instruction. Defined values are:

<table>
<thead>
<tr>
<th>API</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0b0000$</td>
<td>Address Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented.</td>
</tr>
<tr>
<td>$0b0001$</td>
<td>Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC() and HaveEnhancedPAC2() functions returning FALSE.</td>
</tr>
<tr>
<td>$0b0010$</td>
<td>Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE, and the HaveEnhancedPAC2() function returning FALSE.</td>
</tr>
<tr>
<td>$0b0011$</td>
<td>Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE.</td>
</tr>
<tr>
<td>$0b0100$</td>
<td>Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE.</td>
</tr>
<tr>
<td>$0b0101$</td>
<td>Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE.</td>
</tr>
</tbody>
</table>
All other values are reserved.

ARMv8.3-PAuth implements the functionality added by the values 0b0000, 0b0001, and 0b0010.

ARMv8.3-PAuth2 implements the functionality added by the value 0b0011.

ARMv8.3-FPAC implements the functionality added by the values 0b0100 and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of ID_AA64ISAR1_EL1.API is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.

APA, bits [7:4]

From Armv8.3:

Indicates whether QARMA or Architected algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the PACGA instruction. Defined values are:

<table>
<thead>
<tr>
<th>APA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Address Authentication using an Architected algorithm is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Address Authentication using the QARMA algorithm is implemented, with the HaveEnhancedPAC() and</td>
</tr>
<tr>
<td></td>
<td>HaveEnhancedPAC2() functions returning FALSE.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Address Authentication using the QARMA algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE and the HaveEnhancedPAC2() function returning FALSE.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Address Authentication using the QARMA algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning FALSE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Address Authentication using the QARMA algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Address Authentication using the QARMA algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.3-PAuth implements the functionality added by the values 0b0000, 0b0001, and 0b0010.

ARMv8.3-PAuth2 implements the functionality added by the value 0b0011.

ARMv8.3-FPAC implements the functionality added by the values 0b0100 and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of the ID_AA64ISAR1_EL1.API is non-zero, this field must have the value 0b0000.

Otherwise:

Reserved, RES0.
**DPB, bits [3:0]**

**From Armv8.2:**

Data Persistence writeback. Indicates support for the DC CVAP and DC CVADP instructions in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>DPB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>DC CVAP not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>DC CVAP supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>DC CVAP and DC CVADP supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-DCPoP implements the functionality identified by the value 0b0001.

ARMv8.2-DCCVADP implements the functionality identified by the value 0b0010.

From Armv8.2 to Armv8.4, the only permitted value is 0b0001.

From Armv8.5, the only permitted value is 0b0010

**Otherwise:**

Reserved, RES0.

If ID_AA64ISAR1_EL1.{API, APA} == {0, 0}, then:

- The TCR_EL1.{TBID,TBID0}, TCR_EL2.{TBID0,TBID1}, TCR_EL2.TBID and TCR_EL3.TBID bits are RES0.
- APIAKeyHi_EL1, APIAKeyLo_EL1, APIBKeyHi_EL1, APIBKeyLo_EL1, APDAKeyHi_EL1, APDAKeyLo_EL1, APDBKeyHi_EL1, APDBKeyLo_EL1 are not allocated.
- SCTLR_ELx.EnIA, SCTLR_ELx.EnIB, SCTLR_ELx.EnDA, SCTLR_ELx.EnDB are all RES0.

If ID_AA64ISAR1_EL1.{GPI, GPA, API, APA} == {0, 0, 0, 0}, then:

- HCR_EL2.APK and HCR_EL2.API are RES0.
- SCR_EL3.APK and SCR_EL3.API are RES0.

**Accessing the ID_AA64ISAR1_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ID_AA64ISAR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
if IsFeatureImplemented("ARMv8.4-IDST") then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
else
AArch64.SystemAccessTrap(EL1, 0x18);
else
UNDEFINED;
elsif PSTATE.EL == EL1 then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
else
return ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL2 then
return ID_AA64ISAR1_EL1;
elsif PSTATE.EL == EL3 then
return ID_AA64ISAR1_EL1;
The ID_AA64MMFR0_EL1 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64MMFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64MMFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECV</td>
<td>FGT</td>
<td>RES0</td>
<td>ExS</td>
<td>TGran4</td>
<td>TGran64</td>
<td>TGran16</td>
<td>BigEndEL0</td>
<td>SNSMem</td>
<td>BigEnd</td>
<td>ASIDBits</td>
<td>PARange</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**ECV, bits [63:60]**

**From Armv8.6:**

Presence of Enhanced Counter Virtualization. Defined values are:

<table>
<thead>
<tr>
<th>ECV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Enhanced Counter Virtualization is not implemented. Supports</td>
</tr>
<tr>
<td></td>
<td>CNTHTCTL_EL2, (ECV, EL1TVT, EL1TVCT, EL1NVPCT, EL1NVVCT, EVNTIS),</td>
</tr>
<tr>
<td></td>
<td>CNTKCTL_EL1, EVNTIS, CNTPCTSS_EL0 counter views, and</td>
</tr>
<tr>
<td></td>
<td>CNTVCTSS_EL0 counter views. Extends the PMSR_EL1, PCT, PMSR_EL2, PCT,</td>
</tr>
<tr>
<td></td>
<td>TRFCR_EL1, TS, and TRFCR_EL2, TS fields.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, and also includes support for CNTPOFF_EL2.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-ECV implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.6, the only permitted values are 0b0001 and 0b0010.

**Otherwise:**

Reserved, RES0.

**FGT, bits [59:56]**

**From Armv8.6:**

Presence of the Fine-Grained Trap controls:
• HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGRT0_EL2, HFGITR_EL2 and HFGWTR_EL2 registers, and their associated traps.
• MDCR_EL2 TDCC and MDCR_EL3.TDCC.
• SCR_EL3.FGTEn.

Defined values are:

<table>
<thead>
<tr>
<th>FGT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The fine-grained trap controls are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The fine-grained trap controls are implemented, and can generate</td>
</tr>
<tr>
<td></td>
<td>fine-grained traps of EL1 and EL0 functionality.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-FGT implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

Bits [55:48]

Reserved, RES0.

ExS, bits [47:44]

From Armv8.5:

Support for disabling context synchronizing exception entry and exit. Defined values are:

<table>
<thead>
<tr>
<th>ExS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>All exception entries and exits are context synchronization events.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Non-context synchronizing exception entry and exit are supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.5-CSEH implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

TGran4_2, bits [43:40]

From Armv8.5:

Support for 4KB memory granule size for stage 2. Defined values are:

<table>
<thead>
<tr>
<th>TGran4_2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>4KB Stage 2 granule is identified in the TGran4 field</td>
</tr>
<tr>
<td>0b0001</td>
<td>4KB granule not supported at stage 2</td>
</tr>
<tr>
<td>0b0010</td>
<td>4KB granule supported at stage 2</td>
</tr>
</tbody>
</table>

All other values are reserved.

The 0b0000 value is deprecated.

Otherwise:

Reserved, RES0.
**TGran64, bits [39:36]**

*From Arm v8.5:*

Support for 64KB memory granule size for stage 2. Defined values are:

<table>
<thead>
<tr>
<th>TGran64_2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>64KB Stage 2 granule is identified in the TGran64 field</td>
</tr>
<tr>
<td>0b0001</td>
<td>64KB granule not supported at stage 2</td>
</tr>
<tr>
<td>0b0010</td>
<td>64KB granule supported at stage 2</td>
</tr>
</tbody>
</table>

All other values are reserved.

The 0b0000 value is deprecated.

**Otherwise:**

Reserved, RES0.

---

**TGran16, bits [35:32]**

*From Arm v8.5:*

Support for 16KB memory granule size for stage 2. Defined values are:

<table>
<thead>
<tr>
<th>TGran16_2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>16KB Stage 2 granule is identified in the TGran16 field</td>
</tr>
<tr>
<td>0b0001</td>
<td>16KB granule not supported at stage 2</td>
</tr>
<tr>
<td>0b0010</td>
<td>16KB granule supported at stage 2</td>
</tr>
</tbody>
</table>

All other values are reserved.

The 0b0000 value is deprecated.

**Otherwise:**

Reserved, RES0.

---

**TGran4, bits [31:28]**

Support for 4KB memory translation granule size. Defined values are:

<table>
<thead>
<tr>
<th>TGran4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>4KB granule supported.</td>
</tr>
<tr>
<td>0b1111</td>
<td>4KB granule not supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

---

**TGran64, bits [27:24]**

Support for 64KB memory translation granule size. Defined values are:

<table>
<thead>
<tr>
<th>TGran64</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>64KB granule supported.</td>
</tr>
<tr>
<td>0b1111</td>
<td>64KB granule not supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

---

**TGran16, bits [23:20]**

Support for 16KB memory translation granule size. Defined values are:
<table>
<thead>
<tr>
<th>TGran16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>16KB granule not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>16KB granule supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**BigEndEL0, bits [19:16]**

Mixed-endian support at EL0 only. Defined values are:

<table>
<thead>
<tr>
<th>BigEndEL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No mixed-endian support at EL0. The <code>SCTLR_EL1.E0E</code> bit has a fixed value.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Mixed-endian support at EL0. The <code>SCTLR_EL1.E0E</code> bit can be configured.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is invalid and is RES0 if the BigEnd field, bits [11:8], is not 0b0000.

**SNSMem, bits [15:12]**

Secure versus Non-secure Memory distinction. Defined values are:

<table>
<thead>
<tr>
<th>SNSMem</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Does not support a distinction between Secure and Non-secure Memory.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Does support a distinction between Secure and Non-secure Memory.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**BigEnd, bits [11:8]**

Mixed-endian configuration support. Defined values are:

<table>
<thead>
<tr>
<th>BigEnd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No mixed-endian support. The <code>SCTLR_ELx.EE</code> bits have a fixed value.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Mixed-endian support. The <code>SCTLR_ELx.EE</code> and <code>SCTLR_EL1.E0E</code> bits can be configured.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**ASIDBits, bits [7:4]**

Number of ASID bits. Defined values are:

<table>
<thead>
<tr>
<th>ASIDBits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>8 bits.</td>
</tr>
<tr>
<td>0b0010</td>
<td>16 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**PARange, bits [3:0]**

Physical Address range supported. Defined values are:
<table>
<thead>
<tr>
<th>PARange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>32 bits, 4GB.</td>
</tr>
<tr>
<td>0b0001</td>
<td>36 bits, 64GB.</td>
</tr>
<tr>
<td>0b0010</td>
<td>40 bits, 1TB.</td>
</tr>
<tr>
<td>0b0011</td>
<td>42 bits, 4TB.</td>
</tr>
<tr>
<td>0b0100</td>
<td>44 bits, 16TB.</td>
</tr>
<tr>
<td>0b0101</td>
<td>48 bits, 256TB.</td>
</tr>
<tr>
<td>0b0110</td>
<td>52 bits, 4PB.</td>
</tr>
</tbody>
</table>

All other values are reserved.

The value 0b0110 is permitted only if the implementation includes ARMv8.2-LPA, otherwise it is reserved.

**Accessing the ID_AA64MMFR0_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64MMFR0_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL2 then
  return ID_AA64MMFR0_EL1;
elsif PSTATE.EL == EL3 then
  return ID_AA64MMFR0_EL1;
```
The ID_AA64MMFR1_EL1 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64MMFR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64MMFR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ETS | TWED |
| XNX | SpecSEI | PAN | LO | HPDS | VH | VMID Bits | HAFDBS |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:40]**

Reserved, RES0.

**ETS, bits [39:36]**

*From Armv8.0:*

Support for Enhanced Translation Synchronization. Defined values are:

<table>
<thead>
<tr>
<th>ETS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Enhanced Translation Synchronization is not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Enhanced Translation Synchronization is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.0-ETS implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

**Otherwise:**

Reserved, RES0.

**TWED, bits [35:32]**

*From Armv8.6:*

Support for the configurable delayed trapping of WFE. Defined values are:
Meaning

<table>
<thead>
<tr>
<th>TWED</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Configurable delayed trapping of WFE is not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Configurable delayed trapping of WFE is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-TWED implements the functionality identified by the value 0b0001.

From Armv8.6, the permitted values are 0b0000 and 0b0001.

Otherwise:

Reserved, RES0.

**XNX, bits [31:28]**

From Armv8.2:

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

<table>
<thead>
<tr>
<th>XNX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Distinction between EL0 and EL1 execute-never control at stage 2 not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Distinction between EL0 and EL1 execute-never control at stage 2 supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-TTS2UXN implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

**SpecSEI, bits [27:24]**

When RAS is implemented:

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

<table>
<thead>
<tr>
<th>SpecSEI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The PE never generates an SError interrupt due to an External abort on a speculative read.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The PE might generate an SError interrupt due to an External abort on a speculative read.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0. This provides no information about whether the PE generates a speculative SError interrupt.

**PAN, bits [23:20]**

From Armv8.1:

Privileged Access Never. Indicates support for the PAN bit in PSTATE, SPSR_EL1, SPSR_EL2, SPSR_EL3, and DSPSR_EL0. Defined values are:
<table>
<thead>
<tr>
<th>PAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PAN not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PAN supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>PAN supported and <strong>AT S1E1RP</strong> and <strong>AT S1E1WP</strong> instructions supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-PAN implements the functionality identified by the value 0b0001.

ARMv8.2-ATS1E1 implements the functionality added by the value 0b0010.

In Armv8.1, the only permitted value is 0b0001.

From Armv8.2, the only permitted value is 0b0010.

**Otherwise:**

Reserved, RES0.

**LO, bits [19:16]**

**From Armv8.1:**

LORegions. Indicates support for LORegions. Defined values are:

<table>
<thead>
<tr>
<th>LO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>LORegions not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>LORegions supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-LOR implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

**Otherwise:**

Reserved, RES0.

**HPDS, bits [15:12]**

**From Armv8.1:**

Hierarchical permission disables bits in translation tables. Defined values are:

<table>
<thead>
<tr>
<th>HPDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Disabling of hierarchical controls not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Disabling of hierarchical controls supported with the <strong>TCR_EL1</strong>. {HPD1, HPD0}, <strong>TCR_EL2</strong>. HPD or <strong>TCR_EL2</strong>. {HPD1, HPD0}, and <strong>TCR_EL3</strong>. HPD bits.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-HPD implements the functionality identified by the value 0b0001.

ARMv8.2-TTPBHA implements the functionality identified by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.
Otherwise:

Reserved, RES0.

**VH, bits [11:8]**

**From Armv8.1:**

Virtualization Host Extensions. Defined values are:

<table>
<thead>
<tr>
<th>VH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Virtualization Host Extensions not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Virtualization Host Extensions supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-VHE implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

**VMIDBits, bits [7:4]**

**From Armv8.1:**

Number of VMID bits. Defined values are:

<table>
<thead>
<tr>
<th>VMIDBits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>8 bits</td>
</tr>
<tr>
<td>0b0010</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-VMID16 implements the functionality identified by the value 0b0010.

From Armv8.1, the permitted values are 0b0000 and 0b0010.

Otherwise:

Reserved, RES0.

**HAFDBS, bits [3:0]**

**From Armv8.1:**

Hardware updates to Access flag and Dirty state in translation tables. Defined values are:

<table>
<thead>
<tr>
<th>HAFDBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Hardware update of the Access flag and dirty state are not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Hardware update of the Access flag is supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Hardware update of both the Access flag and dirty state is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-TTHM implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.
Otherwise:

Reserved, RES0.

**Accessing the ID_AA64MMFR1_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ID_AA64MMFR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
if IsFeatureImplemented("ARMv8.4-IDST") then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_AA64MMFR1_EL1;
elsif PSTATE.EL == EL2 then
  return ID_AA64MMFR1_EL1;
elsif PSTATE.EL == EL3 then
  return ID_AA64MMFR1_EL1;
ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2

The ID_AA64MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers see Principles of the ID scheme for fields in ID registers.

Configuration

This register is present only from Armv8.2. Otherwise, direct accesses to ID_AA64MMFR2_EL1 are RES0 from EL1, EL2, and EL3.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ID_AA64MMFR2_EL1 is a 64-bit register.

Field descriptions

The ID_AA64MMFR2_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| E0PD | EVT | BBM | TTL | RES0 | FWB | IDS | AT |
| ST | NV | CCIDX | VARange | IESB | LSM | UAO | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

E0PD, bits [63:60]

From Armv8.5:

Indicates support for the E0PD mechanism. Defined values are:

<table>
<thead>
<tr>
<th>E0PD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>E0PDx mechanism is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>E0PDx mechanism is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.5-E0PD implements the functionality identified by the value 0b0001.

In Armv8.4, the permitted values are 0b0000 and 0b001.

From Armv8.5, the only permitted values is 0b0001.

Otherwise:

Reserved, RES0.
EVT, bits [59:56]

From Armv8.2:

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR_EL2.\{TTLBOS, TTLBIS, TOCU, TICAB, TID4\} traps. Defined values are:

<table>
<thead>
<tr>
<th>EVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>HCR_EL2.{TOCU, TICAB, TID4} traps are supported. HCR_EL2.{TTLBOS, TTLBIS} traps are not supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>HCR_EL2.{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARmv8.2-EVT implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.1, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0010 when EL2 is implemented.

Otherwise:

Reserved, RES0.

BBM, bits [55:52]

From Armv8.4:

Allows identification of the requirements of the hardware to have break-before-make sequences when changing block size for a translation.

<table>
<thead>
<tr>
<th>BBM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Level 0 support for changing block size is supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Level 1 support for changing block size is supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Level 2 support for changing block size is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARmv8.4-TTRem implements the functionality identified by the values 0b0000, 0b0001, and 0b0010.

From Armv8.4, the permitted values are 0b0000, 0b0001, and 0b0010.

Otherwise:

Reserved, RES0.

TTL, bits [51:48]

From Armv8.4:

Indicates support for TTL field in address operations. Defined values are:

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>TLB maintenance instructions by address have bits[47:44] as RES0.</td>
</tr>
<tr>
<td>0b0001</td>
<td>TLB maintenance instructions by address have bits[47:44] holding the TTL field.</td>
</tr>
</tbody>
</table>

All other values are reserved.
ARMv8.4-TTL implements the functionality identified by the value 0b0001.

This field affects TLBI IPAS2E1, TLBI IPAS2E1IS, TLBI IPAS2E1OS, TLBI IPAS2LE1, TLBI IPAS2LE1IS, TLBI IPAS2LE1OS, TLBI VAAE1, TLBI VAAE1IS, TLBI VAAE1OS, TLBI VALE1, TLBI VALE1IS, TLBI VALE1OS, TLBI VAE1, TLBI VAE1IS, TLBI VAE1OS, TLBI VAE2, TLBI VAE2IS, TLBI VAE2OS, TLBI VAE3, TLBI VAE3IS, TLBI VAE3OS, TLBI VALE1, TLBI VALE1IS, TLBI VALE1OS, TLBI VALE2, TLBI VALE2IS, TLBI VALE2OS, TLBI VALE3, TLBI VALE3IS, TLBI VALE3OS.

From Armv8.4, the only permitted value is 0b0001.

Otherwise:
Reserved, RES0.

Bits [47:44]
Reserved, RES0.

FWB, bits [43:40]

From Armv8.4:

Indicates support for HCR_EL2.FWB

<table>
<thead>
<tr>
<th>FWB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>HCR_EL2.FWB bit is not supported and the field is RES0</td>
</tr>
<tr>
<td>0b0001</td>
<td>HCR_EL2.FWB is supported.</td>
</tr>
</tbody>
</table>

If ARMv8.4-SecEL2 is implemented the only permitted value is 0b0001

All other values reserved.

This field resets to an architecturally unknown value.

Otherwise:
Reserved, RES0.

IDS, bits [39:36]

From Armv8.4:

Indicates the value of ESR_ELx.EC that reports an exception generated by a read access to the feature ID space.

Defined values are:

<table>
<thead>
<tr>
<th>IDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>An exception which is generated by a read access to the feature</td>
</tr>
<tr>
<td></td>
<td>ID space, other than a trap caused by HCR_EL2.TIDx, SCTLR_EL1.UCT, or</td>
</tr>
<tr>
<td></td>
<td>SCTLR_EL2.UCT, is reported by ESR_ELx.EC == 0x0.</td>
</tr>
<tr>
<td>0b0001</td>
<td>All exceptions generated by an AArch64 read access to the feature ID</td>
</tr>
<tr>
<td></td>
<td>space are reported by ESR_ELx.EC == 0x18.</td>
</tr>
</tbody>
</table>

All other values are reserved.

The Feature ID space is defined as the System register space in AArch64 with op0==3, op1=={0, 1, 3}, CRn==0, CRm=={0-7}, op2=={0-7}.

ARMv8.4-IDST implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.
**Otherwise:**
Reserved, RES0.

**AT, bits [35:32]**

From Armv8.4:

Identifies support for unaligned single-copy atomicity and atomic functions. Defined values are:

<table>
<thead>
<tr>
<th>AT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Unaligned single-copy atomicity and atomic functions are not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Unaligned single-copy atomicity and atomic functions with a 16-byte address range aligned to 16-bytes are supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-LSE implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

**Otherwise:**
Reserved, RES0.

**ST, bits [31:28]**

From Armv8.4:

Identifies support for small translation tables. Defined values are:

<table>
<thead>
<tr>
<th>ST</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 39.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 48 for 4KB and 16KB granules, and 47 for 64KB granules.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-TTST implements the functionality identified by the value 0b0001.

If ARMv8.4-SecEL2 is implemented the only permitted value is 0b0001.

In an implementation which does not support Secure EL2, the permitted values are 0b0000 and 0b0001.

**Otherwise:**
Reserved, RES0.

**NV, bits [27:24]**

From Armv8.3:

Nested Virtualization. If EL2 is implemented, indicates support for the use of nested virtualization. Defined values are:

<table>
<thead>
<tr>
<th>NV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Nested virtualization is not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The HCR_EL2.NV, HCR_EL2.NV1, HCR_EL2.AT bits are implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The VNCR_EL2 register and the HCR_EL2.{AT, NV, NV1, NV2} bits are implemented.</td>
</tr>
</tbody>
</table>
All other values are reserved.

In Armv8.2, the only permitted value is \texttt{0b0000}.

In Armv8.3, the permitted values are:

\begin{itemize}
  \item When EL2 is not implemented, \texttt{0b0000}.
  \item When EL2 is implemented, \texttt{0b0001}.
\end{itemize}

The feature ARMv8.3-NV implements the functionality identified by the value \texttt{0b0001}.

In Armv8.4, the permitted values are:

\begin{itemize}
  \item When EL2 is not implemented, \texttt{0b0000}.
  \item When EL2 is implemented, \texttt{0b0010}.
\end{itemize}

The feature ARMv8.4-NV implements the functionality identified by the value \texttt{0b0010}.

\textbf{Otherwise:}

Reserved, \texttt{RES0}.

\textbf{CCIDX, bits [23:20]}

\textbf{From Armv8.3:}

Support for the use of revised \texttt{CCSIDR_EL1} register format. Defined values are:

\begin{center}
\begin{tabular}{|c|c|}
  \hline
  CCIDX & Meaning \tabularnewline
  \hline
  \texttt{0b0000} & 32-bit format implemented for all levels of the CCSIDR_EL1. \tabularnewline
  \texttt{0b0001} & 64-bit format implemented for all levels of the CCSIDR_EL1. \tabularnewline
  \hline
\end{tabular}
\end{center}

All other values are reserved.

This feature is identified as ARMv8.3-CCIDX.

From Armv8.3, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

\textbf{Otherwise:}

Reserved, \texttt{RES0}.

\textbf{VARange, bits [19:16]}

\textbf{From Armv8.2:}

Indicates support for a larger virtual address. Defined values are:

\begin{center}
\begin{tabular}{|c|c|}
  \hline
  VARange & Meaning \tabularnewline
  \hline
  \texttt{0b0000} & VMSAv8-64 supports 48-bit VAs. \tabularnewline
  \texttt{0b0001} & VMSAv8-64 supports 52-bit VAs when using the 64KB translation granule. The other translation granules support 48-bit VAs. \tabularnewline
  \hline
\end{tabular}
\end{center}

All other values are reserved.

ARMv8.2-LVA implements the functionality identified by the value \texttt{0b0001}.

From Armv8.2, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

\textbf{Otherwise:}

Reserved, \texttt{RES0}. 

Page 948
IESB, bits [15:12]

From Armv8.2:

Indicates support for the IESB bit in the SCTLR_ELx registers. Defined values are:

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>IESB bit in the SCTLR_ELx registers is not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>IESB bit in the SCTLR_ELx registers is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-IESB implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

LSM, bits [11:8]

From Armv8.2:

Indicates support for LSMAOE and nTLSMD bits in SCTLR_EL1 and SCTLR_EL2. Defined values are:

<table>
<thead>
<tr>
<th>LSM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>LSMAOE and nTLSMD bits not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>LSMAOE and nTLSMD bits supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-LSMAOC implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

UAO, bits [7:4]

From Armv8.2:

User Access Override. Defined values are:

<table>
<thead>
<tr>
<th>UAO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>UAO not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>UAO supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-UAO implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

CnP, bits [3:0]

From Armv8.2:

Common not Private translations. Defined values are:
<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Common not Private translations not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Common not Private translations supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

**Accessing the ID_AA64MMFR2_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64MMFR2_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_AA64MMFR2_EL1) || boolean IMPLEMENTATION_DEFINED "ID_AA64MMFR2 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64MMFR2_EL1;
        elsif PSTATE.EL == EL2 then
            return ID_AA64MMFR2_EL1;
        elsif PSTATE.EL == EL3 then
            return ID_AA64MMFR2_EL1;
```
The ID_AA64PFR0_EL1 characteristics are:

**Purpose**

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

The external register EDPFR gives information from this register.

**Attributes**

ID_AA64PFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64PFR0_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CSV3 | CSV2 | RES0 | DIT | AMU | MPAM | SEL2 | SVE |
| RAS | GIC | AdvSIMD | FP | EL3 | EL2 | EL1 | EL0 |

**CSV3, bits [63:60]**

From Armv8.5:

Speculative use of faulting data. Defined values are:

<table>
<thead>
<tr>
<th>CSV3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>This Device does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by instructions newer than the load in the speculative sequence</td>
</tr>
<tr>
<td>0b0001</td>
<td>Data loaded under speculation with a permission or domain fault cannot be used to form an address or generate condition codes or SVE predicate values to be used by instructions newer than the load in the speculative sequence</td>
</tr>
</tbody>
</table>

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

**Otherwise:**

Reserved, RES0.
CSV2, bits [59:56]

From Armv8.5:

Speculative use of out of context branch targets. Defined values are:

<table>
<thead>
<tr>
<th>CSV2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>This Device does not disclose whether branch targets trained in one hardware described context can affect speculative execution in a different hardware described context.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Branch targets trained in one hardware described context can only affect speculative execution in a different hardware described context in a hard-to-determine way. Contexts do not include the SCXTNUM_ELx register contexts, and these registers are not supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Branch targets trained in one hardware described context can only affect speculative execution in a different hardware described context in a hard-to-determine way. Contexts include the SCXTNUM_ELx register contexts, and these registers are supported.</td>
</tr>
</tbody>
</table>

From Armv8.5 the only permitted values are 0b0001 or 0b0010.
All other values are reserved.

Otherwise:

Reserved, RES0.

Bits [55:52]

Reserved, RES0.

DIT, bits [51:48]

From Armv8.4:

Data Independent Timing. Defined values are:

<table>
<thead>
<tr>
<th>DIT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>AArch64 does not guarantee constant execution time of any instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AArch64 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.
ARMv8.4-DIT implements the functionality identified by the value 0b0001.
From Armv8.4, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

AMU, bits [47:44]

From Armv8.4:

Activity Monitors Extension. Defined values are:
<table>
<thead>
<tr>
<th>AMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Activity Monitors Extension is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AMUv1 for Armv8.4 is implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>AMUv1 for Armv8.6 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.</td>
</tr>
</tbody>
</table>

All other values are reserved.

AMUv1 implements the functionality identified by the value 0b0001.

ARMv8.6-AMU implements the functionality identified by the value 0b0010.

Otherwise:

Reserved, RES0.

**MPAM, bits [43:40]**

*From Armv8.2:*

MPAM Extension. Defined values are:

<table>
<thead>
<tr>
<th>MPAM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>MPAM is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>MPAM is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**SEL2, bits [39:36]**

*From Armv8.4:*

Secure EL2. Defined values are:

<table>
<thead>
<tr>
<th>SEL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Secure EL2 is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Secure EL2 is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**SVE, bits [35:32]**

*From Armv8.2:*

Scalable Vector Extension. Defined values are:

<table>
<thead>
<tr>
<th>SVE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE architectural state and programmers’ model are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE architectural state and programmers’ model are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If implemented, refer to [ID_AA64ZFR0_EL1](#) for information about which SVE instructions are available.
Otherwise:

Reserved, RES0.

**RAS, bits [31:28]**

RAS Extension version. The defined values of this field are:

<table>
<thead>
<tr>
<th>RAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No RAS Extension.</td>
</tr>
<tr>
<td>0b0001</td>
<td>RAS Extension present.</td>
</tr>
<tr>
<td>0b0010</td>
<td>ARMv8.4-RAS present. As 0b0001, and adds support for:</td>
</tr>
<tr>
<td></td>
<td>• If EL3 is implemented, ARMv8.4-DFE.</td>
</tr>
<tr>
<td></td>
<td>• Additional ERXMISC&lt;m&gt; EL1 System registers.</td>
</tr>
<tr>
<td></td>
<td>• Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and the</td>
</tr>
<tr>
<td></td>
<td>SCRL_EL3 FIEN and HCR_EL2 FIEN trap controls, to support the optional</td>
</tr>
<tr>
<td></td>
<td>RAS Common Fault Injection Model Extension.</td>
</tr>
<tr>
<td></td>
<td>Error records accessed through System registers conform to RAS System</td>
</tr>
<tr>
<td></td>
<td>Architecture v1.1, which includes simplifications to ERR&lt;n&gt;STATUS and</td>
</tr>
<tr>
<td></td>
<td>support for the optional RAS Timestamp and RAS Common Fault Injection</td>
</tr>
<tr>
<td></td>
<td>Model Extensions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.4, when ARMv8.4-DFE is not implemented, and ERRDR_EL1.NUM is zero, the permitted values are IMPLEMENTATION DEFINED 0b0001 or 0b0010. Otherwise from Armv8.4 the only permitted value is 0b0010.

ARMv8.4-RAS implements the functionality identified by the value 0b0010.

In Armv8.2, the only permitted value is 0b0001.

In Armv8.1 and Armv8.0, the permitted values are 0b0000 and 0b0001.

**GIC, bits [27:24]**

System register GIC interface support. Defined values are:

<table>
<thead>
<tr>
<th>GIC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>GIC CPU interface system registers not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.</td>
</tr>
<tr>
<td>0b0011</td>
<td>System register interface to version 4.1 of the GIC CPU interface is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**AdvSIMD, bits [23:20]**

Advanced SIMD. Defined values are:

<table>
<thead>
<tr>
<th>AdvSIMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Advanced SIMD is implemented, including support for the following SISD and SIMD operations:</td>
</tr>
<tr>
<td></td>
<td>• Integer byte, halfword, word and doubleword element operations.</td>
</tr>
<tr>
<td></td>
<td>• Single-precision and double-precision floating-point arithmetic.</td>
</tr>
<tr>
<td></td>
<td>• Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As for 0b0000, and also includes support for half-precision floating-point arithmetic.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Advanced SIMD is not implemented.</td>
</tr>
</tbody>
</table>
All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- **0b0000** in an implementation with Advanced SIMD support that does not include the ARMv8.2-FP16 extension.
- **0b0001** in an implementation with Advanced SIMD support that includes the ARMv8.2-FP16 extension.
- **0b1111** in an implementation without Advanced SIMD support.

**FP, bits [19:16]**

Floating-point. Defined values are:

<table>
<thead>
<tr>
<th>FP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Floating-point is implemented, and includes support for:</td>
</tr>
<tr>
<td></td>
<td>• Single-precision and double-precision floating-point types.</td>
</tr>
<tr>
<td></td>
<td>• Conversions between single-precision and half-precision data types,</td>
</tr>
<tr>
<td></td>
<td>• double-precision and half-precision data types.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As for 0b0000, and also includes support for half-precision floating-</td>
</tr>
<tr>
<td></td>
<td>point arithmetic.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Floating-point is not implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- **0b0000** in an implementation with floating-point support that does not include the ARMv8.2-FP16 extension.
- **0b0001** in an implementation with floating-point support that includes the ARMv8.2-FP16 extension.
- **0b1111** in an implementation without floating-point support.

**EL3, bits [15:12]**

EL3 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL3 is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL3 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL3 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**EL2, bits [11:8]**

EL2 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL2 is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL2 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL2 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**EL1, bits [7:4]**

EL1 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>EL1 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL1 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

All other values are reserved.
EL0, bits [3:0]

EL0 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>EL0 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL0 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the ID_AA64PFR0_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ID_AA64PFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64PFR0_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_AA64PFR0_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_AA64PFR0_EL1;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The ID_AA64PFR1_EL1 characteristics are:

**Purpose**

Reserved for future expansion of information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

**Attributes**

ID_AA64PFR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64PFR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>RES0</th>
<th>MPAM_frac</th>
<th>RAS_frac</th>
<th>MTE</th>
<th>SSBS</th>
<th>BT</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:20]**

Reserved, RES0.

**MPAM_frac, bits [19:16]**

From Armv8.6:

MPAM Extension fractional field.

<table>
<thead>
<tr>
<th>MPAM_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If \texttt{ID-AA64PFR0_EL1.MPAM == 0b0000}, MPAM Extension not implemented. If \texttt{ID-AA64PFR0_EL1.MPAM == 0b0001}, MPAM Extension version 1.0 is implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>If \texttt{ID-AA64PFR0_EL1.MPAM == 0b0000}, implements MPAM version 0.1 which is like version 1.1 but reduces support for Secure PARTIDs. If \texttt{ID-AA64PFR0_EL1.MPAM == 0b0001}, implements MPAM extension version 1.1 and adds support for TIDR bit in MPAM2_EL2 to provide trapping of \texttt{MPAMIDR_EL1} when \texttt{MPAMHCR_EL2} is not present.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Otherwise:**

Reserved, RES0.

**RAS_frac, bits [15:12]**
From Armv8.4:

RAS Extension fractional field.

<table>
<thead>
<tr>
<th>RAS_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If ID_AA64PFR0_EL1.RAS == 0b0001, RAS Extension implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>If ID_AA64PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for:</td>
</tr>
<tr>
<td></td>
<td>• Additional ERXMISC&lt;m&gt; EL1 System registers.</td>
</tr>
<tr>
<td></td>
<td>• Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension.</td>
</tr>
<tr>
<td></td>
<td>Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR&lt;n&gt;STATUS, and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is valid only if ID_AA64PFR0_EL1.RAS == 0b0001.

Otherwise:

Reserved, RES0.

MTE, bits [11:8]

From Armv8.5:

Support for the Memory Tagging Extension.

<table>
<thead>
<tr>
<th>MTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Memory Tagging Extension is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Memory Tagging Extension instructions accessible at EL0 are implemented. Instructions and System Registers defined by the extension not configurably accessible at EL0 are Unallocated and other System Register fields defined by the extension are RES0.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Memory Tagging Extension is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.5-MemTag implements the functionality identified by the value 0b0001.

When ID_AA64PFR1_EL1.MTE != 0b0010:

• All register fields added to existing System registers and Special-purpose registers as part of the extension are RES0, and treated as 0.

• The following System registers are UNDEFINED:
  ◦ GMID_EL1, GCR_EL1, RGSR_EL1, TFSRE0_EL1, and TFSR_ELx.

• The following System instructions are UNDEFINED:
  ◦ DC CGSW, DC CIGSW, DC IGSW, DC CGDSW, DC CIGDSW, DC IGDSW, DC IGVAC, and DC IGDVAC.

• The following instructions are UNDEFINED:
  ◦ LDGM, STGM, and STZGM.

• The Tagged memory type encoding in MAIR_ELx is UNPREDICTABLE.
Otherwise:

Reserved, RES0.

**SSBS, bits [7:4]**

*From Armv8.5:*

Speculative Store Bypassing controls in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>AArch64 provides no mechanism to control the use of Speculative Store Bypassing.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AArch64 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.</td>
</tr>
<tr>
<td>0b0010</td>
<td>AArch64 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypassing Safe, and the MSR and MRS instructions to directly read and write the PSTATE.SSBS field.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**BT, bits [3:0]**

*From Armv8.5:*

Branch Target Identification mechanism support in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>BT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The Branch Target Identification mechanism is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The Branch Target Identification mechanism is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.5-BTI implements the functionality identified by the value 0b0001.

From Armv8.5, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

**Accessing the ID_AA64PFR1_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_AA64PFR1_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64PFR1_EL1;
    end
elsif PSTATE.EL == EL2 then
    return ID_AA64PFR1_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AA64PFR1_EL1;
else
    return ID_AA64PFR1_EL1;
end
ID_AA64ZFR0_EL1, SVE Feature ID register 0

The ID_AA64ZFR0_EL1 characteristics are:

**Purpose**

Provides additional information about the implemented features of the AArch64 Scalable Vector Extension, when the ID_AA64PFR0_EL1.SVE field is not zero.

For general information about the interpretation of the ID registers see Principles of the ID scheme for fields in ID registers.

**Configuration**

This register is present only when SVE is implemented. Otherwise, direct accesses to ID_AA64ZFR0_EL1 are RAZ.

**Attributes**

ID_AA64ZFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AA64ZFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:60</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>59:56</td>
<td>F64MM, bits [59:56]</td>
</tr>
<tr>
<td>55:52</td>
<td>From Armv8.2:</td>
</tr>
<tr>
<td>51</td>
<td>Indicates support for SVE FP64 double-precision floating-point matrix multiplication instructions. Defined values are:</td>
</tr>
<tr>
<td>50</td>
<td>F64MM</td>
</tr>
<tr>
<td>49</td>
<td>Meaning</td>
</tr>
<tr>
<td>48</td>
<td>0b0000</td>
</tr>
<tr>
<td>47</td>
<td>0b0001</td>
</tr>
<tr>
<td>45:40</td>
<td>All other values are reserved.</td>
</tr>
<tr>
<td>39</td>
<td>ARMv8.2-F64MM implements the functionality identified by 0b0001.</td>
</tr>
<tr>
<td>38:36</td>
<td>Otherwise:</td>
</tr>
<tr>
<td>35:32</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

---

Page 961
F32MM, bits [55:52]

From Armv8.2:

Indicates support for the SVE FP32 single-precision floating-point matrix multiplication instruction. Defined values are:

<table>
<thead>
<tr>
<th>F32MM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>FP32 matrix multiplication instruction is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FMMLA instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-F32MM implements the functionality identified by 0b0001.

Otherwise:

Reserved, RES0.

Bits [51:48]

Reserved, RES0.

I8MM, bits [47:44]

From Armv8.2:

Indicates support for SVE Int8 matrix multiplication instructions. Defined values are:

<table>
<thead>
<tr>
<th>I8MM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Int8 matrix multiplication instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SMMLA, SUDOT, UMMLA, USMMLA, and USDOT instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-I8MM implements the functionality identified by 0b0001.

From Armv8.6, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

SM4, bits [43:40]

When SVE2 is implemented:

Indicates support for SVE2 SM4 instructions. Defined values are:

<table>
<thead>
<tr>
<th>SM4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE2 SM4 instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE2 SM4E and SM4EKEY instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.
Bits [39:36]  
Reserved, RES0.

SHA3, bits [35:32]  
When SVE2 is implemented:

Indicates support for the SVE2 SHA-3 instruction. Defined values are:

<table>
<thead>
<tr>
<th>SHA3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE2 SHA-3 instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE2 RAX1 instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:
Reserved, RES0.

Bits [31:24]  
Reserved, RES0.

BF16, bits [23:20]  
From Armv8.2:
Indicates support for SVE BFloat16 instructions. Defined values are:

<table>
<thead>
<tr>
<th>BF16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>BFloat16 instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>BFCVT, BFCVTNT, BF DOT, BFMLALB, BFMLALT, and BFMM MLA instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-BF16 implements the functionality identified by 0b0001.
From Armv8.6, the only permitted value is 0b0001.

Otherwise:
Reserved, RES0.

BitPerm, bits [19:16]  
When SVE2 is implemented:
Indicates support for SVE2 bit permute instructions. Defined values are:

<table>
<thead>
<tr>
<th>BitPerm</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE2 bit permute instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE2 BDEP, BEXT and BGRP instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:
Reserved, RES0.
Bits [15:8]

Reserved, RES0.

AES, bits [7:4]

When SVE2 is implemented:

Indicates support for SVE2 AES instructions. Defined values are:

<table>
<thead>
<tr>
<th>AES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE2 AES instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE2 AESE, AESD, AESMC and AESIMC instructions are implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, plus SVE2 PMULLB and PMULLT instructions with 64-bit source.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

SVEver, bits [3:0]

Scalable Vector Extension instruction set version. Defined values are:

<table>
<thead>
<tr>
<th>SVEver</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE and the non-optional SVE2 instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved. This field is only valid if the ID_AA64PFR0_EL1.SVE field is not zero.

Accessing the ID_AA64ZFR0_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_AA64ZFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  else if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_AA64ZFR0_EL1) || boolean IMPLEMENTATION_DEFINED "ID_AA64ZFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return ID_AA64ZFR0_EL1;
  else if PSTATE.EL == EL2 then
    return ID_AA64ZFR0_EL1;
  else if PSTATE.EL == EL3 then
    return ID_AA64ZFR0_EL1;
ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0

The ID_AFR0_EL1 characteristics are:

**Purpose**

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, MIDR_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_AFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_AFR0[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_AFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_AFR0_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | IMPLEMENTATION DEFINED | IMPLEMENTATION DEFINED | IMPLEMENTATION DEFINED | IMPLEMENTATION DEFINED |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:16]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [15:12]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [11:8]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [7:4]**

IMPLEMENTATION DEFINED.

**IMPLEMENTATION DEFINED, bits [3:0]**

IMPLEMENTATION DEFINED.
Accessing the ID_AFR0_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_AFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end;
    else
        UNDEFINED;
    end;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AFR0_EL1;
    end;
elsif PSTATE.EL == EL2 then
    return ID_AFR0_EL1;
elsif PSTATE.EL == EL3 then
    return ID_AFR0_EL1;
The ID_DFR0_EL1 characteristics are:

### Purpose

Provides top level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, MIDR_EL1.

For general information about the interpretation of the ID registers see ‘Principles of the ID scheme for fields in ID registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

### Configuration

AArch64 System register ID_DFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_DFR0[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

### Attributes

ID_DFR0_EL1 is a 64-bit register.

### Field descriptions

The ID_DFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:32]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved, RES0</td>
<td></td>
</tr>
</tbody>
</table>

#### TraceFilt, bits [31:28]

Armv8.4 Self-hosted Trace Extension version. Defined values are:

<table>
<thead>
<tr>
<th>TraceFilt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Armv8.4 Self-hosted Trace Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Armv8.4 Self-hosted Trace Extension implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-Trace implements the functionality added by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

#### PerfMon, bits [27:24]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the Alternative ID scheme described in ‘Alternative ID scheme used for the Performance Monitors Extension version’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.1.4.
Defined values are:

<table>
<thead>
<tr>
<th>PerfMon</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Performance Monitors Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Performance Monitors Extension version 1 implemented, PMUv1.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Performance Monitors Extension version 2 implemented, PMUv2.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Performance Monitors Extension version 3 implemented, PMUv3.</td>
</tr>
<tr>
<td>0b0100</td>
<td>PMUv3 for Armv8.1. As 0b0011, and also includes support for:</td>
</tr>
<tr>
<td></td>
<td>• Extended 16-bit PMEVTYPER&lt;n&gt;.evtCount field.</td>
</tr>
<tr>
<td></td>
<td>• If EL2 is implemented, the HDCR.HPMD control bit.</td>
</tr>
<tr>
<td>0b0101</td>
<td>PMUv3 for Armv8.4. As 0b0100 and also includes support for the PMMR register.</td>
</tr>
<tr>
<td>0b0110</td>
<td>PMUv3 for Armv8.5. As 0b0101 and also includes support for:</td>
</tr>
<tr>
<td></td>
<td>• 64-bit event counters.</td>
</tr>
<tr>
<td></td>
<td>• If EL2 is implemented, the HDCR.HCCD control bit.</td>
</tr>
<tr>
<td></td>
<td>• If EL3 is implemented, the SPCR.SCCD control bit.</td>
</tr>
<tr>
<td>0b1111</td>
<td>IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value in new implementations.</td>
</tr>
</tbody>
</table>

ARMv8.1-PMU implements the functionality added by the value 0b0100.

ARMv8.4-PMU implements the functionality added by the value 0b0101.

ARMv8.5-PMU implements the functionality added by the value 0b0110.

All other values are reserved.

In any Armv8 implementation, the values 0b0001 and 0b0010 are not permitted.

From Armv8.1, the value 0b0011 is not permitted.

From Armv8.4, the value 0b0100 is not permitted.

From Armv8.5, the value 0b0101 is not permitted.

**Note**

In Armv7, the value 0b0000 can mean that PMUv1 is implemented. PMUv1 is not permitted in an Armv8 implementation.

**MProfDbg, bits [23:20]**

M Profile Debug. Support for memory-mapped debug model for M profile processors. Defined values are:

<table>
<thead>
<tr>
<th>MProfDbg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for M profile Debug architecture, with memory-mapped access.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

**MMapTrc, bits [19:16]**

Memory Mapped Trace. Support for memory-mapped trace model. Defined values are:

<table>
<thead>
<tr>
<th>MMapTrc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Arm trace architecture, with memory-mapped access.</td>
</tr>
</tbody>
</table>
All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

See the ETM Architecture Specification for more information.

**CopTrc, bits [15:12]**

Support for System registers-based trace model, using registers in the coproc == 0b1110 encoding space. Defined values are:

<table>
<thead>
<tr>
<th>CopTrc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Arm trace architecture, with System registers access.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

See the ETM Architecture Specification for more information.

**MMapDbg, bits [11:8]**

Memory Mapped Debug. Support for v7 memory-mapped debug model, for A and R profile processors.

In Armv8-A, this field is RES0.

The optional memory map defined by Armv8 is not compatible with Armv7.

**CopSDbg, bits [7:4]**

Support for a System registers-based Secure debug model, using registers in the coproc = 0b1110 encoding space, for an A profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

**CopDbg, bits [3:0]**

Support for System registers-based debug model, using registers in the coproc == 0b1110 encoding space, for A and R profile processors. Defined values are:

<table>
<thead>
<tr>
<th>CopDbg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for Armv6, v6 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for Armv6, v6.1 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Support for Armv7, v7 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Support for Armv7, v7.1 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Support for Armv8 debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Support for Armv8 debug architecture, with System registers access, and Virtualization Host extensions.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Support for Armv8.2 debug architecture.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Support for Armv8.4 debug architecture.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In any Armv8 implementation, the values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted.

If Armv8.1-VHE is not implemented, the only permitted value is 0b0110.

In an Armv8.0 implementation, the value 0b1000 is not permitted.
Accessing the **ID_DFR0_EL1**

Accesses to this register use the following encodings:

\[
\text{MRS } <\text{Xt}>, \text{ ID_DFR0_EL1}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL0 then}
\]
\[
\text{if IsFeatureImplemented("ARMv8.4-IDST") then}
\]
\[
\text{if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HCR_EL2.TGE == '1' then}
\]
\[
\text{AArch64.SystemAccessTrap(EL2, 0x18);}
\]
\[
\text{else}
\]
\[
\text{AArch64.SystemAccessTrap(EL1, 0x18);}
\]
\[
\text{else}
\]
\[
\text{UNDEFINED;}
\]
\[
\text{elsif PSTATE.EL == EL1 then}
\]
\[
\text{if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HCR_EL2.TID3 == '1' then}
\]
\[
\text{AArch64.SystemAccessTrap(EL2, 0x18);}
\]
\[
\text{else}
\]
\[
\text{return ID_DFR0_EL1;}
\]
\[
\text{elsif PSTATE.EL == EL2 then}
\]
\[
\text{return ID_DFR0_EL1;}
\]
\[
\text{elsif PSTATE.EL == EL3 then}
\]
\[
\text{return ID_DFR0_EL1;}
\]
ID_DFR1_EL1, Debug Feature Register 1

The ID_DFR1_EL1 characteristics are:

**Purpose**

Provides top level information about the debug system in AArch32.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_DFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_DFR1[31:0].

This register is present only from Armv8.6. Otherwise, direct accesses to ID_DFR1_EL1 are RES0.

**Attributes**

ID_DFR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_DFR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:4]**

Reserved, RES0.

**MTPMU, bits [3:0]**

From Armv8.6:

Multi-threaded PMU extension. Defined values are:

<table>
<thead>
<tr>
<th>MTPMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ARMv8.6-MTPMU not implemented. If PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether</td>
</tr>
<tr>
<td></td>
<td>PMEVTYPER&lt;n&gt;_EL0.MT are read/write or RES0.</td>
</tr>
<tr>
<td>0b0001</td>
<td>ARMv8.6-MTPMU implemented and PMEVTYPER&lt;n&gt;_EL0.MT are read/write. When ARMv8.6-MTPMU is</td>
</tr>
<tr>
<td></td>
<td>disabled, the Effective values of PMEVTYPER&lt;n&gt;_EL0.MT are 0.</td>
</tr>
<tr>
<td>0b1111</td>
<td>ARMv8.6-MTPMU not implemented. If PMUv3 is implemented, PMEVTYPER&lt;n&gt;_EL0.MT are RES0.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-MTPMU implements the functionality identified by the value 0b0001.

In an Armv8.6-compliant implementation that includes PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include PMUv3, the value 0b0001 is not permitted.
Otherwise:
Reserved, RES0.

Accessing the ID_DFR1_EL1

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ ID_DFR1_EL1}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_DFR1_EL1) || boolean IMPLEMENTATION_DEFINED "ID_DFR1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_DFR1_EL1;
elsif PSTATE.EL == EL2 then
  return ID_DFR1_EL1;
elsif PSTATE.EL == EL3 then
  return ID_DFR1_EL1;
The ID_ISAR0_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D7.1.3.

**Configuration**

AArch64 System register ID_ISAR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR0[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_ISAR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>RES0</th>
<th>Divide</th>
<th>Debug</th>
<th>Coproc</th>
<th>CmpBranch</th>
<th>BitField</th>
<th>BitCount</th>
<th>Swap</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>36</td>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:28]**

Reserved, RES0.

**Divide, bits [27:24]**

Indicates the implemented Divide instructions. Defined values are:

<table>
<thead>
<tr>
<th>Divide</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0001</td>
<td>Adds SDIV and UDIV in the T32 instruction set.</td>
</tr>
<tr>
<td>0010</td>
<td>As for 0001, and adds SDIV and UDIV in the A32 instruction set.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

**Debug, bits [23:20]**

Indicates the implemented Debug instructions. Defined values are:
### Debug

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds BKPT.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

### Coproc, bits [19:16]

Indicates the implemented System register access instructions. Defined values are:

<table>
<thead>
<tr>
<th>Coproc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds generic CDP, LDC, MCR, MRC, and STC.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and adds generic MCRR and MRRC.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds generic MCRR2 and MRRC2.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

### CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>CmpBranch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds CBNZ and CBZ.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

### BitField, bits [11:8]

Indicates the implemented BitField instructions. Defined values are:

<table>
<thead>
<tr>
<th>BitField</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds BFC, BFI, SBFX, and UBFX.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

### BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions. Defined values are:

<table>
<thead>
<tr>
<th>BitCount</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds CLZ.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

### Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set. Defined values are:
<table>
<thead>
<tr>
<th>Swap</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds SWP and SWPB.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

**Accessing the ID_ISAR0_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, ID_ISAR0_EL1**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b00</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end;
    else
        UNDEFINED;
    end;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR0_EL1;
    end;
elsif PSTATE.EL == EL2 then
    return ID_ISAR0_EL1;
elsif PSTATE.EL == EL3 then
    return ID_ISAR0_EL1;
end;
```
ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1

The ID_ISAR1_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D7.1.3.

**Configuration**

AArch64 System register ID_ISAR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR1[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_ISAR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Jazelle | Interwork | Immediate | IfThen | Extend | Except_AR | Except | Endian |

**Bits [63:32]**

Reserved, RES0.

**Jazelle, bits [31:28]**

Indicates the implemented Jazelle extension instructions. Defined values are:

<table>
<thead>
<tr>
<th>Jazelle</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No support for Jazelle.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the BXJ instruction and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**Interwork, bits [27:24]**

Indicates the implemented Interworking instructions. Defined values are:
## Interwork, bits [5:0]

<table>
<thead>
<tr>
<th>Interwork</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the BX instruction, and the T bit in the PSR.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

## Immediate, bits [23:20]

Indicates the implemented data-processing instructions with long immediates. Defined values are:

<table>
<thead>
<tr>
<th>Immediate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
</tbody>
</table>
| 0b0001    | Adds:  
|           | • The MOVT instruction.  
|           | • The MOV instruction encodings with zero-extended 16-bit immediates.  
|           | • The T32 ADD and SUB instruction encodings with zero-extended 12-bit immediates, and the other ADD, ADR, and SUB encodings cross-referenced by the pseudocode for those encodings. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

## IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>IfThen</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the IT instructions, and the IT bits in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

## Extend, bits [15:12]

Indicates the implemented Extend instructions. Defined values are:

<table>
<thead>
<tr>
<th>Extend</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No scalar sign-extend or zero-extend instructions are implemented, where scalar instructions means non-Advanced SIMD instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SXTB, SXTH, UXTB, and UXTH instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16, SXTAH, UXTB16, UXTAB, UXTAB16, and UXTHA instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

## Except_AR, bits [11:8]

Indicates the implemented A and R profile exception-handling instructions. Defined values are:
Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>Except</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Endian, bits [3:0]

Indicates the implemented Endian instructions. Defined values are:

<table>
<thead>
<tr>
<th>Endian</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SETEND instruction, and the E bit in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

**Accessing the ID_ISAR1_EL1**

Accesses to this register use the following encodings:

```plaintext
MRS <Xt>, ID_ISAR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_ISAR1_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_ISAR1_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_ISAR1_EL1;
```
**ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2**

The ID_ISAR2_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_ISAR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR2][31:0].

In an implementation that supports only AArch64 state, this register is **UNKNOWN**.

**Attributes**

ID_ISAR2_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR2_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Reversal</th>
<th>PSR_AR</th>
<th>MultU</th>
<th>MultS</th>
<th>Mult</th>
<th>MultiAccessInt</th>
<th>MemHint</th>
<th>LoadStore</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, **RES0**.

**Reversal, bits [31:28]**

Indicates the implemented Reversal instructions. Defined values are:

<table>
<thead>
<tr>
<th>Reversal</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the REV, REV16, and REVSH instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the RBIT instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**PSR_AR, bits [27:24]**

Indicates the implemented A and R profile instructions to manipulate the PSR. Defined values are:
<table>
<thead>
<tr>
<th>PSR_AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the SUBS PC,LR,#N instruction.

**MultU, bits [23:20]**

Indicates the implemented advanced unsigned Multiply instructions. Defined values are:

<table>
<thead>
<tr>
<th>MultU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the UMULL and UMLAL instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the UMAAL instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**MultS, bits [19:16]**

Indicates the implemented advanced signed Multiply instructions. Defined values are:

<table>
<thead>
<tr>
<th>MultS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SMULL and SMLAL instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the SMLAB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLATT, SMLAWB, SMLAWT, SMLBB, SMLBT, SMLTB, SMULBB, SMUBLT, SMULT, SMULWB, and SMULWT instructions. Also adds the Q bit in the PSRs.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and adds the SMLAD, SMLADX, SMLAD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSLDX, SMLLA, SMLAR, SMLLS, SMLLSR, SMMLSR, SMUL, SMULR, SMUAD, SMUADX, SMUSD, and SMUSDX instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0011.

**Mult, bits [15:12]**

Indicates the implemented additional Multiply instructions. Defined values are:

<table>
<thead>
<tr>
<th>Mult</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No additional instructions implemented. This means only MUL is implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the MLA instruction.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the MLS instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**MultiAccessInt, bits [11:8]**

Indicates the support for interruptible multi-access instructions. Defined values are:
### MultiAccessInt

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No support. This means the LDM and STM instructions are not interruptible.</td>
</tr>
<tr>
<td>0b0001</td>
<td>LDM and STM instructions are restartable.</td>
</tr>
<tr>
<td>0b0010</td>
<td>LDM and STM instructions are continuable.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

### MemHint, bits [7:4]

Indicates the implemented Memory Hint instructions. Defined values are:

<table>
<thead>
<tr>
<th>MemHint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the PLD instruction.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Adds the PLD instruction. (0b0001 and 0b0010 have identical effects.)</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0001 (or 0b0010), and adds the PLI instruction.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds the PLDW instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0100.

### LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions. Defined values are:

<table>
<thead>
<tr>
<th>LoadStore</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No additional load/store instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the LDRD and STRD instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

### Accessing the ID_ISAR2_EL1

Accesses to this register use the following encodings:

\[
\text{MRS <Xt>, ID_ISAR2_EL1}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR2_EL1;
elsif PSTATE.EL == EL2 then
    return ID_ISAR2_EL1;
elsif PSTATE.EL == EL3 then
    return ID_ISAR2_EL1;
The ID_ISAR3_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR4_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D7.1.3.

**Configuration**

AArch64 System register ID_ISAR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR3[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_ISAR3_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR3_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31-28</td>
<td>Indicates the implemented T32EE instructions. Defined values are:</td>
</tr>
<tr>
<td></td>
<td>T32EE</td>
</tr>
<tr>
<td>31-24</td>
<td>T32EE</td>
</tr>
<tr>
<td>27-24</td>
<td>T32EE</td>
</tr>
<tr>
<td>23-20</td>
<td>T32EE</td>
</tr>
<tr>
<td>19-16</td>
<td>T32EE</td>
</tr>
<tr>
<td>15-12</td>
<td>T32EE</td>
</tr>
<tr>
<td>11-8</td>
<td>T32EE</td>
</tr>
<tr>
<td>7-4</td>
<td>T32EE</td>
</tr>
<tr>
<td>3-0</td>
<td>T32EE</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**T32EE, bits [31:28]**

Indicates the implemented T32EE instructions. Defined values are:

<table>
<thead>
<tr>
<th>T32EE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

**TrueNOP, bits [27:24]**

Indicates the implemented true NOP instructions. Defined values are:
<table>
<thead>
<tr>
<th>TrueNOP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. This means there are no NOP instructions that do not have any register dependencies.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**T32Copy, bits [23:20]**

Indicates the support for T32 non flag-setting MOV instructions. Defined values are:

<table>
<thead>
<tr>
<th>T32Copy</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported. This means that in the T32 instruction set, encoding T1 of the MOV (register) instruction does not support a copy from a low register to a low register.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds support for T32 instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**TabBranch, bits [19:16]**

Indicates the implemented Table Branch instructions in the T32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>TabBranch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the TBB and TBH instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**SynchPrim, bits [15:12]**

Used in conjunction with ID_ISAR3_EL1.AArch32 Instruction Set Attribute Register 3 to indicate the implemented Synchronization Primitive instructions. Defined values are:

<table>
<thead>
<tr>
<th>SynchPrim</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If SynchPrim frac == 0b0000, no Synchronization Primitives implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>If SynchPrim frac == 0b0000, adds the LDREX and STREX instructions. If SynchPrim frac == 0b0011, also adds the CLREX, LDREXB, STREXB, and STREXH instructions. If SynchPrim frac == 0b0000, as for [0b0001, 0b0011] and also adds the LDREXD and STREXD instructions.</td>
</tr>
</tbody>
</table>

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0010.

**SVC, bits [11:8]**

Indicates the implemented SVC instructions. Defined values are:

<table>
<thead>
<tr>
<th>SVC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SVC instruction.</td>
</tr>
</tbody>
</table>
All other values are reserved.

In Armv8-A, the only permitted value is \( 0b0001 \).

**SIMD, bits [7:4]**

Indicates the implemented SIMD instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SSAT and USAT instructions, and the Q bit in the PSRs.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0001, and adds the PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSA, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHAX, SHSUB16, SHSUB8, SHAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHAX, UHSUB16, UHUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQASX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports floating-point and Advanced SIMD instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

**Saturate, bits [3:0]**

Indicates the implemented Saturate instructions. Defined values are:

<table>
<thead>
<tr>
<th>Saturate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. This means no non-Advanced SIMD saturate instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**Accessing the ID_ISAR3_EL1**

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ ID_ISAR3_EL1}
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b000 & 0b0000 & 0b0010 & 0b011 \\
\hline
\end{array}
\]
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR3_EL1;
    end
elsif PSTATE.EL == EL2 then
    return ID_ISAR3_EL1;
elsif PSTATE.EL == EL3 then
    return ID_ISAR3_EL1;
**ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4**

The ID_ISAR4_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D7.1.3.

**Configuration**

AArch64 System register ID_ISAR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR4[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_ISAR4_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR4_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31-28</td>
<td>SWP_frac</td>
</tr>
<tr>
<td>27-24</td>
<td>PSR_M</td>
</tr>
<tr>
<td>23-20</td>
<td>SynchPrim_frac</td>
</tr>
<tr>
<td>19-16</td>
<td>Barrier</td>
</tr>
<tr>
<td>15-12</td>
<td>SMC</td>
</tr>
<tr>
<td>11-8</td>
<td>Writeback</td>
</tr>
<tr>
<td>7-4</td>
<td>WithShifts</td>
</tr>
<tr>
<td>3-0</td>
<td>Unpriv</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**SWP_frac, bits [31:28]**

Indicates support for the memory system locking the bus for SWP or SWPB instructions. Defined values are:

<table>
<thead>
<tr>
<th>SWP_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SWP or SWPB instructions not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SWP or SWPB implemented but only in a uniprocessor context. SWP and SWPB do not guarantee whether memory accesses from other masters can come between the load memory access and the store memory access of the SWP or SWPB.</td>
</tr>
</tbody>
</table>

All other values are reserved. This field is valid only if the ID_ISAR0_Swap_instrs field is 0b0000.

In Armv8-A, the only permitted value is 0b0000.

**PSR_M, bits [27:24]**

Indicates the implemented M profile instructions to modify the PSRs. Defined values are:
<table>
<thead>
<tr>
<th>PSR_M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the M profile forms of the CPS, MRS, and MSR instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

**SynchPrim_frac, bits [23:20]**

Used in conjunction with ID_ISAR3. SynchPrim to indicate the implemented Synchronization Primitive instructions. Possible values are:

<table>
<thead>
<tr>
<th>SynchPrim_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If SynchPrim == 0b0000, no Synchronization Primitives implemented. If</td>
</tr>
<tr>
<td></td>
<td>SynchPrim == 0b0001, adds the LDREX and STREX instructions. If</td>
</tr>
<tr>
<td></td>
<td>SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB,</td>
</tr>
<tr>
<td></td>
<td>STREXH, LDREXD, and STREXD instructions.</td>
</tr>
<tr>
<td>0b0011</td>
<td>If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH,</td>
</tr>
<tr>
<td></td>
<td>STREXH, STREXB, and STREXH instructions.</td>
</tr>
</tbody>
</table>

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0000.

**Barrier, bits [19:16]**

Indicates the implemented Barrier instructions in the A32 and T32 instruction sets. Defined values are:

<table>
<thead>
<tr>
<th>Barrier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. Barrier operations are provided only as System</td>
</tr>
<tr>
<td></td>
<td>instructions in the (coproc==0b1111) encoding space.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the DMB, DSB, and ISB barrier instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**SMC, bits [15:12]**

Indicates the implemented SMC instructions. Defined values are:

<table>
<thead>
<tr>
<th>SMC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SMC instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0001 and 0b0000.

If EL1 cannot use AArch32, then this field has the value 0b0000.

**Writeback, bits [11:8]**

Indicates the support for Writeback addressing modes. Defined values are:

<table>
<thead>
<tr>
<th>Writeback</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions</td>
</tr>
<tr>
<td></td>
<td>support writeback addressing modes. These instructions support all of</td>
</tr>
<tr>
<td></td>
<td>their writeback addressing modes.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds support for all of the writeback addressing modes.</td>
</tr>
</tbody>
</table>

Page 990
All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**WithShifts, bits [7:4]**

Indicates the support for instructions with shifts. Defined values are:

<table>
<thead>
<tr>
<th>WithShifts</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Nonzero shifts supported only in MOV and shift instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds support for shifts of loads and stores over the range LSL 0-3.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0001, and adds support for other constant shift options, both on load/store and other instructions.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds support for register-controlled shift options.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

**Unpriv, bits [3:0]**

Indicates the implemented unprivileged instructions. Defined values are:

<table>
<thead>
<tr>
<th>Unpriv</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. No T variant instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the LDRBT, LDRT, STRBT, and STRT instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

**Accessing the ID_ISAR4_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ID_ISAR4_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return ID_ISAR4_EL1;
  elsif PSTATE.EL == EL2 then
    return ID_ISAR4_EL1;
  elsif PSTATE.EL == EL3 then
    return ID_ISAR4_EL1;
```
ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5

The ID_ISAR5_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, and ID_ISAR4_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D7.1.3.

**Configuration**

AArch64 System register ID_ISAR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR5[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_ISAR5_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR5_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VCMA | RDM | RES0 | CRC32 | SHA2 | SHA1 | AES | SEVL |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**VCMA, bits [31:28]**

From Armv8.3:

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors. Defined values are:

<table>
<thead>
<tr>
<th>VCMA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>The VCMLA and VCADD instructions are not implemented in AArch32.</td>
</tr>
<tr>
<td>00001</td>
<td>The VCMLA and VCADD instructions are implemented in AArch32.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.3-CompNum implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, the only permitted value is 0b0001.
Otherwise:
Reserved, RES0.

**RDM, bits [27:24]**

**From Armv8.1:**

Indicates whether the VQRDMLAH and VQRDMLSH instructions are implemented in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>RDM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No VQRDMLAH and VQRDMLSH instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VQRDMLAH and VQRDMLSH instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-RDMA implements the functionality identified by the value 0b0001.

In Armv8.0, the only permitted value is 0b0000.

From Armv8.1, the only permitted value is 0b0001.

Otherwise:
Reserved, RES0.

**Bits [23:20]**

Reserved, RES0.

**CRC32, bits [19:16]**

Indicates whether the CRC32 instructions are implemented in AArch32 state.

<table>
<thead>
<tr>
<th>CRC32</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No CRC32 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.1, the only permitted value is 0b0001.

**SHA2, bits [15:12]**

Indicates whether the SHA2 instructions are implemented in AArch32 state.

<table>
<thead>
<tr>
<th>SHA2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA2 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

**SHA1, bits [11:8]**

Indicates whether the SHA1 instructions are implemented in AArch32 state.
<table>
<thead>
<tr>
<th>SHA1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA1 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

**AES, bits [7:4]**

Indicates whether the AES instructions are implemented in AArch32 state.

<table>
<thead>
<tr>
<th>AES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No AES instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AESE,AESD,AESMC, and AESIMC implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, plus VMULL (polynomial) instructions operating on 64-bit data quantities.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

**SEVL, bits [3:0]**

Indicates whether the SEVL instruction is implemented in AArch32 state.

<table>
<thead>
<tr>
<th>SEVL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SEVL is implemented as a NOP.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SEVL is implemented as Send Event Local.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

**Accessing the ID_ISAR5_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ID_ISAR5_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_ISAR5_EL1;
  end
elsif PSTATE.EL == EL2 then
  return ID_ISAR5_EL1;
elsif PSTATE.EL == EL3 then
  return ID_ISAR5_EL1;
ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6

The ID_ISAR6_EL1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0_EL1, ID_ISAR1_EL1, ID_ISAR2_EL1, ID_ISAR3_EL1, ID_ISAR4_EL1 and ID_ISAR5_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_ISAR6_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_ISAR6[31:0].

This register is present only from Armv8.2. Otherwise, direct accesses to ID_ISAR6_EL1 are RES0. In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_ISAR6_EL1 is a 64-bit register.

**Field descriptions**

The ID_ISAR6_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | I8MM | BF16 | SPECRES | SB | FHM | DP | JSCVT |

**Bits [63:28]**

Reserved, RES0.

**I8MM, bits [27:24]**

From Armv8.2:

Indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions in AArch32 state. Defined values of this field are:

<table>
<thead>
<tr>
<th>I8MM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Int8 matrix multiplication instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VSMMLA, VSUDOT, VUMMLA, VUSMMLA, and VUSDOT instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-AA32I8MM implements the functionality identified by 0b0001.
Otherwise:

Reserved, RES0.

**BF16, bits [23:20]**

From Armv8.2:

Indicates support for Advanced SIMD and floating-point BFloat16 instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>BF16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>BFloat16 instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VCVT, VCVTB, VCVTT, VDOT, VFMA, and VMMLA instructions with BF16 operand or result types are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARmv8.2-AA32BF16 implements the functionality identified by 0b0001.

Otherwise:

Reserved, RES0.

**SPECRES, bits [19:16]**

Indicates support for Speculation invalidation instruction in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>SPECRES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>CFPRCTX, DVPRCTX, and CPPRCTX instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

**SB, bits [15:12]**

Indicates support for SB instruction in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>SB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SB instruction is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SB instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

**FHM, bits [11:8]**

Indicates support for Advanced SIMD and floating-point VFMAL and VFMSL instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>FHM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>VFMAL and VMFSL instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VFMAL and VMFSL instructions are implemented.</td>
</tr>
</tbody>
</table>

ARmv8.2-FHM implements the functionality identified by the value 0b0001.

In Armv8.0 and Armv8.1, the only permitted value is 0b0000.

In Armv8.2, the permitted values are 0b0000 and 0b0001.
**DP, bits [7:4]**

Indicates the support for dot product instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No dot product instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VUDOT and VSDOT instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-DotProd implements the functionality identified by the value 0b0001.

In Armv8.0 and Armv8.1, the only permitted value is 0b0000.

In Armv8.2, the permitted values are 0b0000 and 0b0001.

**JSCVT, bits [3:0]**

From Armv8.3:

Indicates support for the Javascript conversion instruction in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The VJCVT instruction is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The VJCVT instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.3-JSConv implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, the only permitted value is 0b0001.

**Otherwise:**

Reserved, RES0.

**Accessing the ID_ISAR6_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, ID_ISAR6_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_ISAR6_EL1) || boolean IMPLEMENTATION_DEFINED "ID_ISAR6_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR6_EL1;
elsif PSTATE.EL == EL2 then
    return ID_ISAR6_EL1;
elsif PSTATE.EL == EL3 then
    return ID_ISAR6_EL1;
### ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0

The ID_MMFR0_EL1 characteristics are:

#### Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR1_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

#### Configuration

AArch64 System register ID_MMFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR0[31:0].

In an implementation that supports only AArch64 state, this register is **UNKNOWN**.

#### Attributes

ID_MMFR0_EL1 is a 64-bit register.

#### Field descriptions

The ID_MMFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignments</th>
<th>RES0</th>
<th>InnerShr</th>
<th>FCSE</th>
<th>AuxReg</th>
<th>TCM</th>
<th>ShareLvl</th>
<th>OuterShr</th>
<th>PMSA</th>
<th>VMSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits [31:28]</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>Bits [27:20]</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Bits [19:12]</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Bits [63:32]

Reserved, RES0.

#### InnerShr, bits [31:28]

Innermost Shareability. Indicates the innermost shareability domain implemented. Defined values are:

<table>
<thead>
<tr>
<th>InnerShr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Implemented as Non-cacheable.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented with hardware coherency support.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Shareability ignored.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0_EL1.ShareLvl having the value 0b0001.

When ID_MMFR0_EL1.ShareLvl is zero, this field is UNK.
**FCSE, bits [27:24]**

Indicates whether the implementation includes the FCSE. Defined values are:

<table>
<thead>
<tr>
<th>FCSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for FCSE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

**AuxReg, bits [23:20]**

Auxiliary Registers. Indicates support for Auxiliary registers. Defined values are:

<table>
<thead>
<tr>
<th>AuxReg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Auxiliary Control Register only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0010.

**Note**

Accesses to unimplemented Auxiliary registers are UNDEFINED.

**TCM, bits [19:16]**

Indicates support for TCMs and associated DMAs. Defined values are:

<table>
<thead>
<tr>
<th>TCM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support is IMPLEMENTATION DEFINED. Armv7 requires this setting.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for TCM only, Armv6 implementation.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for TCM and DMA, Armv6 implementation.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**ShareLvl, bits [15:12]**

Shareability Levels. Indicates the number of shareability levels implemented. Defined values are:

<table>
<thead>
<tr>
<th>ShareLvl</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>One level of shareability implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Two levels of shareability implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0001.

**OuterShr, bits [11:8]**

Outermost Shareability. Indicates the outermost shareability domain implemented. Defined values are:
### OuterShr

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Implemented as Non-cacheable.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented with hardware coherency support.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Shareability ignored.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

### PMSA, bits [7:4]

Indicates support for a PMSA. Defined values are:

<table>
<thead>
<tr>
<th>PMSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for IMPLEMENTATION DEFINED PMSA</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for PMSAv6, with a Cache Type Register implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for PMSAv7, with support for memory subsections, Armv7-R profile.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

### VMSA, bits [3:0]

Indicates support for a VMSA. Defined values are:

<table>
<thead>
<tr>
<th>VMSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for IMPLEMENTATION DEFINED VMSA</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for VMSAv6, with Cache and TLB Type Registers implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for VMSAv7, with support for remapping and the Access flag, Armv7-A profile.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As for 0b0100, and adds support for the Long-descriptor translation table format.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0101.

### Accessing the ID_MMFR0_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, ID_MMFR0_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    else
        UNDEFINED;
    end if
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR0_EL1;
    end if
elsif PSTATE.EL == EL2 then
    return ID_MMFR0_EL1;
elsif PSTATE.EL == EL3 then
    return ID_MMFR0_EL1;
end if
ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1

The ID_MMFR1_EL1 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR2_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_MMFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR1[31:0].

In an implementation that supports only AArch64 state, this register is **UNKNOWN**.

**Attributes**

ID_MMFR1_EL1 is a 64-bit register.

**Field descriptions**

The ID_MMFR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BPred | L1TstCln | L1Uni | L1Hvd | L1UniSW | L1HvdSW | L1UniVA | L1HvdVA |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**BPred, bits [31:28]**

Branch Predictor. Indicates branch predictor management requirements. Defined values are:
### L1TstCln, bits [27:24]

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations. Defined values are:

<table>
<thead>
<tr>
<th>L1TstCln</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
</tbody>
</table>
| 0b0001  | Supported Level 1 data cache test and clean operations are:  
          | • Test and clean data cache. |
| 0b0010  | As for 0001, and adds:  
          | • Test, clean, and invalidate data cache. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

### L1Uni, bits [23:20]

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1Uni</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
</tbody>
</table>
| 0b0001 | Supported entire Level 1 cache operations are:  
          | • Invalidate cache, including branch predictor if appropriate.  
          | • Invalidate branch predictor, if appropriate. |
| 0b0010 | As for 0001, and adds:  
          | • Clean cache, using a recursive model that uses the cache dirty status bit.  
          | • Clean and invalidate cache, using a recursive model that uses the cache dirty status bit. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.
L1Hvd, bits [19:16]

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1Hvd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
</tbody>
</table>
| 0b0001 | Supported entire Level 1 cache operations are:  
• Invalidate instruction cache, including branch predictor if appropriate.  
• Invalidate branch predictor, if appropriate. |
| 0b0010 | As for 0001, and adds:  
• Invalidate data cache.  
• Invalidate data cache and instruction cache, including branch predictor if appropriate. |
| 0b0011 | As for 0010, and adds:  
• Clean data cache, using a recursive model that uses the cache dirty status bit.  
• Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

L1UniSW, bits [15:12]

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1UniSW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
</tbody>
</table>
| 0b0001 | Supported Level 1 unified cache line maintenance operations by set/way are:  
• Clean cache line by set/way. |
| 0b0010 | As for 0001, and adds:  
• Clean and invalidate cache line by set/way. |
| 0b0011 | As for 0010, and adds:  
• Invalidate cache line by set/way. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

L1HvdSW, bits [11:8]

Level 1 Harvard cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a Harvard cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1HvdSW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
</tbody>
</table>
| 0b0001 | Supported Level 1 Harvard cache line maintenance operations by set/way are:  
• Clean data cache line by set/way.  
• Clean and invalidate data cache line by set/way. |
| 0b0010 | As for 0001, and adds:  
• Invalidate data cache line by set/way. |
| 0b0011 | As for 0010, and adds:  
• Invalidate instruction cache line by set/way. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.
L1UniVA, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1UniVA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 unified cache line maintenance operations by VA are:</td>
</tr>
<tr>
<td></td>
<td>• Clean cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate cache line by VA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate branch predictor by VA, if branch predictor is</td>
</tr>
<tr>
<td></td>
<td>implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1HvdVA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache line maintenance operations by VA are:</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean instruction cache line by VA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate branch predictor by VA, if branch predictor is</td>
</tr>
<tr>
<td></td>
<td>implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Accessing the ID_MMFR1_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_MMFR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR1_EL1;
elsif PSTATE.EL == EL2 then
    return ID_MMFR1_EL1;
elif PSTATE.EL == EL3 then
    return ID_MMFR1_EL1;
The ID_MMFR2_EL1 characteristics are:

### Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR3_EL1, and ID_MMFR4_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

### Configuration

AArch64 System register ID_MMFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR2[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

### Attributes

ID_MMFR2_EL1 is a 64-bit register.

### Field descriptions

The ID_MMFR2_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>HWAccFlg</td>
<td>Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation. Defined values are:</td>
</tr>
<tr>
<td>61</td>
<td>WFIStall</td>
<td>Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling. Defined values are:</td>
</tr>
<tr>
<td>60</td>
<td>MemBarr</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>UniTLB</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>HvdTLB</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>L1HvdRng</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>L1HvdBG</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>L1HvdFG</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**HWAccFlg, bits [31:28]**

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation. Defined values are:

<table>
<thead>
<tr>
<th>HWAccFlg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for VMSAv7 Access flag, updated in hardware.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

**WFIStall, bits [27:24]**

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling. Defined values are:
### WFISnall

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for WFI stalling.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the permitted values are 0b0000 and 0b0001.

### MemBarr, bits [23:20]

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc==0b1111) encoding space:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported memory barrier System instructions are:</td>
</tr>
<tr>
<td></td>
<td>• Data Synchronization Barrier (DSB).</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Instruction Synchronization Barrier (ISB).</td>
</tr>
<tr>
<td></td>
<td>• Data Memory Barrier (DMB).</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0010.

Arm deprecates the use of these operations. ID_ISAR4.Barrier_instrs indicates the level of support for the preferred barrier instructions.

### UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported unified TLB maintenance operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate all entries in the TLB.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate TLB entry by VA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate TLB entries by ASID match.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0010, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction TLB and data TLB entries by VA All ASID.</td>
</tr>
<tr>
<td></td>
<td>This is a shared unified TLB operation.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0011, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate Hyp mode unified TLB entry by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate entire Non-secure PL1&amp;0 unified TLB.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate entire Hyp mode unified TLB.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As for 0b0100, and adds the following operations:</td>
</tr>
<tr>
<td></td>
<td>TLBIMVALIS, TLBIMVAALIS, TLBIMVALHIS, TLBIMVAL, TLBIMVAAL, TLBIMVALH.</td>
</tr>
<tr>
<td>0b0110</td>
<td>As for 0b0101, and adds the following operations:</td>
</tr>
<tr>
<td></td>
<td>TLBIIIPAS2IS, TLBIIIPAS2LIS, TLBIIIPAS2, TLBIIIPAS2L.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0110.

### HvdTLB, bits [15:12]

If the Unified TLB field (UniTLB, bits [19:16]) is not 0000, then the meaning of this field is IMPLEMENTATION DEFINED. Arm deprecates the use of this field by software.

### L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation. Defined values are:
### L1HvdRng

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache maintenance range operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache range by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

### L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache background fetch operations are:</td>
</tr>
<tr>
<td></td>
<td>• Fetch instruction cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Fetch data cache range by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

### L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache foreground fetch operations are:</td>
</tr>
<tr>
<td></td>
<td>• Fetch instruction cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Fetch data cache range by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

### Accessing the ID.MMFR2_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, ID.MMFR2_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR2_EL1;
elsif PSTATE.EL == EL2 then
    return ID_MMFR2_EL1;
elsif PSTATE.EL == EL3 then
    return ID_MMFR2_EL1;
ID.MMFR3_EL1, AArch32 Memory Model Feature Register 3

The ID.MMFR3_EL1 characteristics are:

### Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID.MMFR0_EL1, ID.MMFR1_EL1, ID.MMFR2_EL1, and ID.MMFR4_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

### Configuration

AArch64 System register ID.MMFR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID.MMFR3[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

### Attributes

ID.MMFR3_EL1 is a 64-bit register.

### Field descriptions

The ID.MMFR3_EL1 bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>Supersec</th>
<th>CMemSz</th>
<th>CohWalk</th>
<th>PAN</th>
<th>MaintBcst</th>
<th>BPMaint</th>
<th>CMaintSW</th>
<th>CMaintVA</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

**Supersec, bits [31:28]**

Supersections. On a VMSA implementation, indicates whether Supersections are supported. Defined values are:

<table>
<thead>
<tr>
<th>Supersec</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Supersections supported.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Supersections not supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b1111.

**CMemSz, bits [27:24]**

Cached Memory Size. Indicates the physical memory size supported by the caches. Defined values are:
CMemSz

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>4GB, corresponding to a 32-bit physical address range.</td>
</tr>
<tr>
<td>0b0001</td>
<td>64GB, corresponding to a 36-bit physical address range.</td>
</tr>
<tr>
<td>0b0010</td>
<td>1TB or more, corresponding to a 40-bit or larger physical address range.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000, 0b0001, and 0b0010.

CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification. Defined values are:

<table>
<thead>
<tr>
<th>CohWalk</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Updates to the translation tables require a clean to the Point</td>
</tr>
<tr>
<td></td>
<td>of Unification to ensure visibility by subsequent translation</td>
</tr>
<tr>
<td></td>
<td>table walks.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Updates to the translation tables do not require a clean to</td>
</tr>
<tr>
<td></td>
<td>the Point of Unification to ensure visibility by subsequent</td>
</tr>
<tr>
<td></td>
<td>translation table walks.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

PAN, bits [19:16]

From Armv8.1:

Privileged Access Never. Indicates support for the PAN bit in CPSR, SPSR, and DSPSR in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>PAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PAN not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PAN supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>PAN supported and ATS1CPRP and ATS1CPWP instructions supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-PAN implements the functionality identified by the value 0b0001.

ARMv8.2-ATS1E1 implements the functionality added by the value 0b0010.

In Armv8.1 the value 0b0000 is not permitted.

From Armv8.2, the only permitted value is 0b0010.

Otherwise:

Reserved, RES0.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast. Defined values are:
MaintBcst

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Cache, TLB, and branch predictor operations only affect local structures.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**BPMaint, bits [11:8]**

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported branch predictor maintenance operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate all branch predictors.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate branch predictors by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**CMaintSW, bits [7:4]**

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported hierarchical cache maintenance instructions by set/way are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache by set/way.</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache by set/way.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache by set/way.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

**CMaintVA, bits [3:0]**

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported hierarchical cache maintenance operations by VA are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate all instruction cache entries.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.
In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

Accessing the ID_MMFR3_EL1

Accesses to this register use the following encodings:

\[ \text{MRS} \ <Xt>, \text{ID_MMFR3_EL1} \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b111</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL0 then}
\]
\[
\text{if IsFeatureImplemented("ARMv8.4-IDST") then}
\]
\[
\text{if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then}
\]
\[
\text{AArch64.SystemAccessTrap(EL2, 0x18);}
\]
\[
\text{else}
\]
\[
\text{AArch64.SystemAccessTrap(EL1, 0x18);}
\]
\[
\text{else}
\]
\[
\text{UNDEFINED;}
\]
\[
\text{elsif PSTATE.EL == EL1 then}
\]
\[
\text{if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then}
\]
\[
\text{AArch64.SystemAccessTrap(EL2, 0x18);}
\]
\[
\text{else}
\]
\[
\text{return ID_MMFR3_EL1;}
\]
\[
\text{elsif PSTATE.EL == EL2 then}
\]
\[
\text{return ID_MMFR3_EL1;}
\]
\[
\text{elsif PSTATE.EL == EL3 then}
\]
\[
\text{return ID_MMFR3_EL1;}
\]
ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4

The ID_MMFR4_EL1 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0_EL1, ID_MMFR1_EL1, ID_MMFR2_EL1, and ID_MMFR3_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_MMFR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR4[31:0].

In an implementation that supports only AArch64 state, this register is **UNKNOWN**.

**Attributes**

ID_MMFR4_EL1 is a 64-bit register.

**Field descriptions**

The ID_MMFR4_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>EVT</td>
</tr>
<tr>
<td>61</td>
<td>EVT</td>
</tr>
<tr>
<td>60</td>
<td>EVT</td>
</tr>
<tr>
<td>59</td>
<td>EVT</td>
</tr>
<tr>
<td>58</td>
<td>EVT</td>
</tr>
<tr>
<td>57</td>
<td>EVT</td>
</tr>
<tr>
<td>56</td>
<td>EVT</td>
</tr>
<tr>
<td>55</td>
<td>EVT</td>
</tr>
<tr>
<td>54</td>
<td>EVT</td>
</tr>
<tr>
<td>53</td>
<td>EVT</td>
</tr>
<tr>
<td>52</td>
<td>EVT</td>
</tr>
<tr>
<td>51</td>
<td>EVT</td>
</tr>
<tr>
<td>50</td>
<td>EVT</td>
</tr>
<tr>
<td>49</td>
<td>EVT</td>
</tr>
<tr>
<td>48</td>
<td>EVT</td>
</tr>
<tr>
<td>47</td>
<td>EVT</td>
</tr>
<tr>
<td>46</td>
<td>EVT</td>
</tr>
<tr>
<td>45</td>
<td>EVT</td>
</tr>
<tr>
<td>44</td>
<td>EVT</td>
</tr>
<tr>
<td>43</td>
<td>EVT</td>
</tr>
<tr>
<td>42</td>
<td>EVT</td>
</tr>
<tr>
<td>41</td>
<td>EVT</td>
</tr>
<tr>
<td>40</td>
<td>EVT</td>
</tr>
<tr>
<td>39</td>
<td>EVT</td>
</tr>
<tr>
<td>38</td>
<td>EVT</td>
</tr>
<tr>
<td>37</td>
<td>EVT</td>
</tr>
<tr>
<td>36</td>
<td>EVT</td>
</tr>
<tr>
<td>35</td>
<td>EVT</td>
</tr>
<tr>
<td>34</td>
<td>EVT</td>
</tr>
<tr>
<td>33</td>
<td>EVT</td>
</tr>
<tr>
<td>32</td>
<td>EVT</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**EVT, bits [31:28]**

When ARMv8.2-EVT is implemented:

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the HCR2.\{TTLBIS, TOCU, TICAB, TID4\} traps. Defined values are:

<table>
<thead>
<tr>
<th>EVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>HCR2.{TOCU, TICAB, TID4} traps are supported.</td>
</tr>
<tr>
<td></td>
<td>HCR2.TTLBIS trap is not supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>HCR2.{TTLBIS, TOCU, TICAB, TID4} traps are supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0, the only permitted value is 0b0000.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5 the permitted values are:
• 0b0000 when EL2 is not implemented.
• 0b0001 when EL2 is implemented. This feature is identified as ARMv8.2-EVT.

Otherwise:
Reserved, RES0.

CCIDX, bits [27:24]
From Armv8.3:
Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated. Defined values are:

<table>
<thead>
<tr>
<th>CCIDX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001. This feature is identified as ARMv8.3-CCIDX.

Otherwise:
Reserved, RAZ.

LSM, bits [23:20]
From Armv8.2:
Indicates support for LSMAOE and nTLSMD bits in HSCTRL and SCTLR. Defined values are:

<table>
<thead>
<tr>
<th>LSM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>LSMAOE and nTLSMD bits not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>LSMAOE and nTLSMD bits supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-LSMAOC implements the functionality identified by the value 0b0001.

Otherwise:
Reserved, RAZ.

HPDS, bits [19:16]
From Armv8.2:
Hierarchical permission disables bits in translation tables. Defined values are:

<table>
<thead>
<tr>
<th>HPDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Disabling of hierarchical controls not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supports disabling of hierarchical controls using the TTBCR2.HPD0, TTBCR2.HPD1, and HTCR.HPD bits.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-AA32HPD implements the functionality identified by the value 0b0001.
ARMv8.2-TTPBHA implements the functionality added by the value 0b0010.

---

**Note**

The value 0b0000 implies that the encoding for TTBCR2 is UNDEFINED.

---

**Otherwise:**

Reserved, RAZ.

**CnP, bits [15:12]**

**From Armv8.2:**

Common not Private translations. Defined values are:

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Common not Private translations not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Common not Private translations supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2 the only permitted value is 0b0001.

**Otherwise:**

Reserved, RAZ.

**XNX, bits [11:8]**

**From Armv8.2:**

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

<table>
<thead>
<tr>
<th>XNX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Distinction between EL0 and EL1 execute-never control at stage 2 not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Distinction between EL0 and EL1 execute-never control at stage 2 supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-TTS2UXN implements the functionality identified by the value 0b0001.

When ARMv8.2-TTS2UXN is implemented:

- If all of the following conditions are true it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4_EL1.XNX is 0b0000 or 0b0001:
  - ID_AA64MMFR1_EL1.XNX == 1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the only permitted value is 0b0001.

**Otherwise:**

Reserved, RAZ.

**AC2, bits [7:4]**

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. Defined values are:
Meaning

0b0000  ACTLR2 and HACTLR2 are not implemented.
0b0001  ACTLR2 and HACTLR2 are implemented.

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

SpecSEI, bits [3:0]

When RAS is implemented:

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

<table>
<thead>
<tr>
<th>SpecSEI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The PE never generates an SError interrupt due to an External abort on a speculative read.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The PE might generate an SError interrupt due to an External abort on a speculative read.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0. This provides no information about whether the PE generates a speculative SError interrupt.

Accessing the ID_MMFR4_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_MMFR4_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4_EL1) || boolean IMPLEMENTATION_DEFINED "ID_MMFR4_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return ID_MMFR4_EL1;
  elsif PSTATE.EL == EL2 then
    return ID_MMFR4_EL1;
  elsif PSTATE.EL == EL3 then
    return ID_MMFR4_EL1;
```
The ID_MMFR5_EL1 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_MMFR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_MMFR5[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_MMFR5_EL1 is a 64-bit register.

**Field descriptions**

The ID_MMFR5_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>61</td>
<td>ETS</td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:4]**

Reserved, RES0.

**ETS, bits [3:0]**

**From Armv8.0:**

Support for Enhanced Translation Synchronization. Defined values are:

<table>
<thead>
<tr>
<th>ETS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Enhanced Translation Synchronization is not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Enhanced Translation Synchronization is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.0-ETS implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

**Otherwise:**

Reserved, RES0.
Accessing the ID_MMFR5_EL1

Accesses to this register use the following encodings:

\[ MRS <Xt>, ID_MMFR5_EL1 \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR5_EL1) || boolean IMPLEMENTATION_DEFINED "ID_MMFR5_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_MMFR5_EL1;
  end
elsif PSTATE.EL == EL2 then
  return ID_MMFR5_EL1;
elsif PSTATE.EL == EL3 then
  return ID_MMFR5_EL1;
ID_PFR0_EL1, AArch32 Processor Feature Register 0

The ID_PFR0_EL1 characteristics are:

**Purpose**

Gives top-level information about the instruction sets supported by the PE in AArch32 state.

Must be interpreted with ID_PFR1_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_PFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR0[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

**Attributes**

ID_PFR0_EL1 is a 64-bit register.

**Field descriptions**

The ID_PFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>RAS</th>
<th>DIT</th>
<th>AMU</th>
<th>CSV2</th>
<th>State3</th>
<th>State2</th>
<th>State1</th>
<th>State0</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
</tr>
<tr>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>36</td>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**RAS, bits [31:28]**

RAS Extension version.

<table>
<thead>
<tr>
<th>RAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No RAS Extension.</td>
</tr>
<tr>
<td>0b0001</td>
<td>RAS Extension present.</td>
</tr>
<tr>
<td>0b0010</td>
<td>ARMv8.4-RAS present. As 0b0001, and adds support for additional ERXMISC&lt;m&gt; System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR&lt;n&gt;STATUS, and support for the optional RAS Timestamp Extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.4, when ARMv8.4-DFE is not implemented, and ERRIDR.NUM is zero, the permitted values are IMPLEMENTATION DEFINED 0b0001 or 0b0010. Otherwise from Armv8.4 the only permitted value is 0b0010.

ARMv8.4-RAS implements the functionality identified by the value 0b0010.

In Armv8.2, the only permitted value is 0b0001.
In Armv8.1 and Armv8.0, the permitted values are 0b0000 and 0b0001.

**DIT, bits [27:24]**

*From Armv8.4:*

Data Independent Timing. Defined values are:

<table>
<thead>
<tr>
<th>DIT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>AArch32 does not guarantee constant execution time of any instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AArch32 provides the CPSR.DIT mechanism to guarantee constant execution time of certain instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

*Otherwise:*

Reserved, RES0.

**AMU, bits [23:20]**

*From Armv8.4:*

Activity Monitors Extension. Defined values are:

<table>
<thead>
<tr>
<th>AMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Activity Monitors Extension is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AMUv1 for Armv8.4 is implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>AMUv1 for Armv8.6 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.</td>
</tr>
</tbody>
</table>

All other values are reserved.

AMUv1 implements the functionality identified by the value 0b0001.

ARMv8.6-AMU implements the functionality identified by the value 0b0010.

*Otherwise:*

Reserved, RES0.

**CSV2, bits [19:16]**

*From Armv8.5:*

Speculative use of out of context branch targets. Defined values are:

<table>
<thead>
<tr>
<th>CSV2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>This Device does not disclose whether branch targets trained in one hardware described context can affect speculative execution in a different hardware described context.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Branch targets trained in one hardware described context can only affect speculative execution in a different hardware described context in a hard-to-determine way.</td>
</tr>
</tbody>
</table>

From Armv8.5 the only permitted value is 0b0001.

All other values are reserved.
Otherwise:

Reserved, RES0.

**State3, bits [15:12]**

T32EE instruction set support. Defined values are:

<table>
<thead>
<tr>
<th>State3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>T32EE instruction set implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**State2, bits [11:8]**

Jazelle extension support. Defined values are:

<table>
<thead>
<tr>
<th>State2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Jazelle extension implemented, without clearing of IOSCR.CV on exception entry.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Jazelle extension implemented, with clearing of IOSCR.CV on exception entry.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**State1, bits [7:4]**

T32 instruction set support. Defined values are:

<table>
<thead>
<tr>
<th>State1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>T32 instruction set not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>T32 encodings before the introduction of Thumb-2 technology implemented:</td>
</tr>
<tr>
<td></td>
<td>• All instructions are 16-bit.</td>
</tr>
<tr>
<td></td>
<td>• A BL or BLX is a pair of 16-bit instructions.</td>
</tr>
<tr>
<td></td>
<td>• 32-bit instructions other than BL and BLX cannot be encoded.</td>
</tr>
<tr>
<td>0b0011</td>
<td>T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0011.

**State0, bits [3:0]**

A32 instruction set support. Defined values are:

<table>
<thead>
<tr>
<th>State0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>A32 instruction set not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>A32 instruction set implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.
Accessing the ID_PFR0_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_PFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_PFR0_EL1;
elsif PSTATE.EL == EL2 then
  return ID_PFR0_EL1;
elsif PSTATE.EL == EL3 then
  return ID_PFR0_EL1;
The ID_PFR1_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

Configuration

AArch64 System register ID_PFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR1[31:0].

In an implementation that supports only AArch64 state, this register is UNKNOWN.

Attributes

ID_PFR1_EL1 is a 64-bit register.

Field descriptions

The ID_PFR1_EL1 bit assignments are:

<p>| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|</p>
<table>
<thead>
<tr>
<th>GIC</th>
<th>Virt_frac</th>
<th>Sec_frac</th>
<th>GenTimer</th>
<th>Virtualization</th>
<th>MProgMod</th>
<th>Security</th>
<th>ProgMod</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

GIC, bits [31:28]

System register GIC CPU interface. Defined values are:

<table>
<thead>
<tr>
<th>GIC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>GIC CPU interface system registers not implemented.</td>
</tr>
<tr>
<td>0001</td>
<td>System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.</td>
</tr>
<tr>
<td>0011</td>
<td>System register interface to version 4.1 of the GIC CPU interface is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Virt_frac, bits [27:24]

Virtualization fractional field. When the Virtualization field is 0b0000, determines the support for features from the ARMv7 Virtualization Extensions. Defined values are:
Virt_frac

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0000</th>
<th>The following features of the ARMv7 Virtualization Extensions are implemented:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• The SCR.SIF bit, if EL3 is implemented.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The MSR (banked register) and MRS (banked register) instructions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The ERET instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:
- 0b0000 when EL2 is implemented.
- 0b0001 when EL2 is not implemented.

This field is only valid when the value of ID_PFR1_EL1.Virtualization is 0,otherwise it holds the value 0b0000.

Note

The ID_ISAR registers do not identify whether the instructions added by the ARMv7 Virtualization Extensions are implemented.

---

Sec_frac, bits [23:20]

Security fractional field. When the Security field is 0b0000, determines the support for features from the ARMv7 Security Extensions. Defined values are:

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0000</th>
<th>No features from the ARMv7 Security Extensions are implemented.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0b0001</td>
<td>The following features from the ARMv7 Security Extensions are implemented:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The VBAR register.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The TTBCR.PD0 and TTBCR.PD1 bits.</td>
</tr>
<tr>
<td></td>
<td>0b0010</td>
<td>As for 0b0001, plus the ability to access Secure or Non-secure physical memory is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:
- 0b0000 when EL3 is implemented.
- 0b0001 or 0b0010 when EL3 is not implemented.

This field is only valid when the value of ID_PFR1_EL1.Security is 0,otherwise it holds the value 0b0000.

GenTimer, bits [19:16]

Generic Timer support. Defined values are:

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0000</th>
<th>Generic Timer is not implemented.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0b0001</td>
<td>Generic Timer is implemented.</td>
</tr>
<tr>
<td></td>
<td>0b0010</td>
<td>Generic Timer is implemented, and also includes support for CNTHCTL.EVTNIS and CNTKCTL.EVTNIS fields, and CNTPTCSS and CNTVCVSS counter views.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-ECV implements the functionality identified by the value 0b0010.

In Armv8.0, Armv8.1, Armv8.2, Armv8.3, Armv8.4, and Armv8.5, the only permitted value is 0b0001.
From Armv8.6, the only permitted value is **0b0010**.

**Virtualization, bits [15:12]**

Virtualization support. Defined values are:

<table>
<thead>
<tr>
<th>Virtualization</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL2, Hyp mode, and the HVC instruction not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL2, Hyp mode, the HVC instruction, and all the features described by Virt_frac == 0b0001 implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

If EL1 cannot use AArch32 then this field has the value 0b0000.

**Note**

The ID_ISARs do not identify whether the HVC instruction is implemented.

**MProgMod, bits [11:8]**

M profile programmers’ model support. Defined values are:

<table>
<thead>
<tr>
<th>MProgMod</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for two-stack programmers' model.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**Security, bits [7:4]**

Security support. Defined values are:

<table>
<thead>
<tr>
<th>Security</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL3, Monitor mode, and the SMC instruction not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL3, Monitor mode, the SMC instruction, and all the features described by Sec_frac == 0b0001 implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the ability to set the NSACR.RFR bit. Not permitted in Armv8 as the NSACR.RFR bit is RES0.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 when EL3 is not implemented.
- 0b0001 when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

If EL1 cannot use AArch32 then this field has the value 0b0000.
ProgMod, bits [3:0]

Support for the standard programmers' model for Armv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes. Defined values are:

<table>
<thead>
<tr>
<th>ProgMod</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0001 and 0b0000.

If EL1 cannot use AArch32 then this field has the value 0b0000.

Accessing the ID_PFR1_EL1

Accesses to this register use the following encodings:

MRS <Xt>, ID_PFR1_EL1

```plaintext
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

```plaintext
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_PFR1_EL1;
elsif PSTATE.EL == EL2 then
  return ID_PFR1_EL1;
elsif PSTATE.EL == EL3 then
  return ID_PFR1_EL1;
```
The ID_PFR2_EL1 characteristics are:

**Purpose**

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0_EL1 and ID_PFR1_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register ID_PFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register ID_PFR2[31:0].

In an implementation that supports only AArch64 state, this register is **UNKNOWN**.

**Attributes**

ID_PFR2_EL1 is a 64-bit register.

**Field descriptions**

The ID_PFR2_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | RAS_frac | SSBS | CSV3 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:12]**

Reserved, RES0.

**RAS_frac, bits [11:8]**

*From Armv8.4:*

RAS Extension fractional field.

<table>
<thead>
<tr>
<th>RAS_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If ID_PFR0_EL1.RAS == 0b0001, RAS Extension implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>If ID_PFR0_EL1.RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC&lt;m&gt; System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR&lt;n&gt;STATUS and support for the optional RAS Timestamp Extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is valid only if ID_PFR0_EL1.RAS == 0b0001.
Otherwise:
Reserved, RES0.

**SSBS, bits [7:4]**

*From Armv8.5:*

Speculative Store Bypassing controls in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>AArch32 provides no mechanism to control the use of Speculative Store Bypassing.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AArch32 provides the PSTATE_SSBS mechanism to mark regions that are Speculative Store Bypass Safe.</td>
</tr>
</tbody>
</table>

From Armv8.0, the permitted values are 0b0000 and 0b0001.
From Armv8.5, the only permitted value is 0b0001.
All other values are reserved.

Otherwise:
Reserved, RES0.

**CSV3, bits [3:0]**

*From Armv8.5:*

Speculative use of faulting data. Defined values are:

<table>
<thead>
<tr>
<th>CSV3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>This Device does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by instructions newer than the load in the speculative sequence</td>
</tr>
<tr>
<td>0b0001</td>
<td>Data loaded under speculation with a permission or domain fault cannot be used to form an address or generate condition codes or SVE predicate values to be used by instructions newer than the load in the speculative sequence</td>
</tr>
</tbody>
</table>

From Armv8.0, the permitted values are 0b0000 and 0b0001.
From Armv8.5, the only permitted value is 0b0001.
All other values are reserved.

Otherwise:
Reserved, RES0.

**Accessing the ID_PFR2_EL1**

Accesses to this register use the following encodings:

\[
\begin{array}{cccccc}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b11 & 0b000 & 0b0000 & 0b0011 & 0b100 \\
\end{array}
\]
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return ID_PFR2_EL1;
elsif PSTATE.EL == EL2 then
  return ID_PFR2_EL1;
elsif PSTATE.EL == EL3 then
  return ID_PFR2_EL1;
IFS32_EL2, Instruction Fault Status Register (EL2)

The IFS32_EL2 characteristics are:

**Purpose**

Allows access to the AArch32 IFSR register from AArch64 state only. Its value has no effect on execution in AArch64 state.

**Configuration**

AArch64 System register IFS32_EL2 bits [31:0] are architecturally mapped to AArch32 System register IFSR[31:0].

If EL1 is AArch64 only, this register is **UNDEFINED**.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not **RES0**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

IFS32_EL2 is a 64-bit register.

**Field descriptions**

The IFS32_EL2 bit assignments are:

**When TTBCR.EAE == 0:**

| 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 | RES0 |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | FnV RES0 ExTR RES0 FS[4] LPAE RES0 FS[3:0] |

**Bits [63:17]**

Reserved, RES0.

**FnV, bit [16]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>IFAR is not valid, and holds an <strong>UNKNOWN</strong> value.</td>
</tr>
</tbody>
</table>

This field is only valid for a synchronous External abort other than a synchronous External abort on a translation table walk. It is **RES0** for all other Prefetch Abort exceptions.

This field resets to an architecturally **UNKNOWN** value.

**Bits [15:13]**

Reserved, RES0.
**ExT, bit [12]**

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This field resets to an architecturally UNKNOWN value.

**Bit [11]**

Reserved, RES0.

**FS[4], bit [10]**

This field is bit[4] of FS[4:0].


<table>
<thead>
<tr>
<th>FS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000001</td>
<td>PC alignment fault.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>0b00011</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b00101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b00110</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b00111</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b01000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b01001</td>
<td>Domain fault, level 1.</td>
</tr>
<tr>
<td>0b01011</td>
<td>Domain fault, level 2.</td>
</tr>
<tr>
<td>0b01100</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b01101</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b01110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b01111</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b10000</td>
<td>TLB conflict abort.</td>
</tr>
<tr>
<td>0b10100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown fault).</td>
</tr>
<tr>
<td>0b11001</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b11100</td>
<td>Synchronous parity or ECC error on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b11110</td>
<td>Synchronous parity or ECC error on translation table walk, level 2.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b11001, 0b11100, and 0b11110 are reserved.

The FS field is split as follows:

- FS[4] is IFSR32_EL2[10].
- FS[3:0] is IFSR32_EL2[3:0].

This field resets to an architecturally UNKNOWN value.

**LPAE, bit [9]**

On taking a Data Abort exception, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table formats.</td>
</tr>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table formats.</td>
</tr>
</tbody>
</table>

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

This field resets to an architecturally UNKNOWN value.
Bits [8:4]

Reserved, RES0.

FS[3:0], bits [3:0]

This field is bits[3:0] of FS[4:0].


When TTBCR.EAE == 1:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>FnV</td>
</tr>
<tr>
<td>29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>ExT</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>LPAE</td>
</tr>
<tr>
<td>25</td>
<td>RES0</td>
</tr>
<tr>
<td>24</td>
<td>STATUS</td>
</tr>
</tbody>
</table>

Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>IFAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

This field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table formats.</td>
</tr>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table formats.</td>
</tr>
</tbody>
</table>

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.
This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{Bits [8:6]}

Reserved, RES0.

\textbf{STATUS, bits [5:0]}

Fault status bits. All encodings not shown below are reserved:

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault in TTBR0 or TTBR1.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b001011</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100001</td>
<td>PC alignment fault.</td>
</tr>
<tr>
<td>0b100010</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011101, 0b011110, and 0b011111, are reserved.

The lookup level associated with a fault is:

- For a fault generated on a translation table walk, the lookup level of the walk being performed.
- For a Translation fault, the lookup level of the translation table that gave the fault. If a fault occurs because a stage of address translation is disabled, or because the input address is outside the range specified by the appropriate base address register or registers, the fault is reported as a fault at level 1.
- For an Access flag fault, the lookup level of the translation table that gave the fault.
- For a Permission fault, including a Permission fault caused by hierarchical permissions, the lookup level of the final level of translation table accessed for the translation. That is, the lookup level of the translation table that returned a Block or Page descriptor.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{Accessing the IFSR32_EL2}

Accesses to this register use the following encodings:
MRS <Xt>, IFSR32_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return IFSR32_EL2;
elsif PSTATE.EL == EL3 then
  return IFSR32_EL2;

MSR IFSR32_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  IFSR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  IFSR32_EL2 = X[t];
The ISR_EL1 characteristics are:

**Purpose**

Shows the pending status of the IRQ, FIQ, or SError interrupt.

When executing at EL2, EL3 or Secure EL1 when `SCR_EL3.EEL2 == 0b0`, this shows the pending status of the physical IRQ, FIQ, or SError interrupts.

When executing at either Non-secure EL1 or at Secure EL1 when `SCR_EL3.EEL2 == 0b1`:

- If the `HCR_EL2.{IMO,FMO,AMO}` bit has a value of 1, the corresponding `ISR_EL1.{I,F,A}` bit shows the pending status of the virtual IRQ, FIQ, or SError.
- If the `HCR_EL2.{IMO,FMO,AMO}` bit has a value of 0, the corresponding `ISR_EL1.{I,F,A}` bit shows the pending status of the physical IRQ, FIQ, or SError.

**Configuration**

AArch64 System register ISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register `ISR[31:0]`.

**Attributes**

ISR_EL1 is a 64-bit register.

**Field descriptions**

The ISR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:9]**

Reserved, RES0.

**A, bit [8]**

SError interrupt pending bit.

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No pending SError.</td>
</tr>
<tr>
<td>0b1</td>
<td>An SError interrupt is pending.</td>
</tr>
</tbody>
</table>

If the SError interrupt is edge-triggered, this field is cleared to zero when the physical SError interrupt is taken.

**I, bit [7]**

IRQ pending bit. Indicates whether an IRQ interrupt is pending:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No pending IRQ.</td>
</tr>
<tr>
<td>0b1</td>
<td>An IRQ interrupt is pending.</td>
</tr>
</tbody>
</table>
**F, bit [6]**

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No pending FIQ.</td>
</tr>
<tr>
<td>0b1</td>
<td>An FIQ interrupt is pending.</td>
</tr>
</tbody>
</table>

**Bits [5:0]**

Reserved, RES0.

**Accessing the ISR_EL1**

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ ISR_EL1}
\]

<table>
<thead>
<tr>
<th></th>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b000</td>
<td></td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.ISR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ISR_EL1;
    elsif PSTATE.EL == EL2 then
        return ISR_EL1;
    elsif PSTATE.EL == EL3 then
        return ISR_EL1;
    endif
endif
The LORC_EL1 characteristics are:

**Purpose**

Enables and disables LORegions, and selects the current LORegion descriptor.

**Configuration**

This register is present only when ARMv8.1-LOR is implemented. Otherwise, direct accesses to LORC_EL1 are UNDEFINED.

If no LORegion descriptors are supported by the PE, then this register is RES0.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

LORC_EL1 is a 64-bit register.

**Field descriptions**

The LORC_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9:2</td>
<td>Descriptor Select. Selects the current LORegion descriptor accessed by LORSA_EL1, LOREA_EL1, and LORN_EL1. The number of LORegion descriptors in IMPLEMENTATION DEFINED. The maximum number of LORegion descriptors supported is 256. If the number is less than 256, then bits[63:M+2] are RES0, where M is Log2(Number of LORegion descriptors supported by the implementation). If this field points to an LORegion descriptor that is not supported by an implementation, then the registers LORN_EL1, LOREA_EL1, and LORSA_EL1 are RES0. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Enable. Indicates whether LORegions are enabled.</td>
</tr>
<tr>
<td>EN</td>
<td>Meaning</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>0b0</td>
<td>Disabled. Memory accesses do not match any LORegions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enabled. Memory accesses may match a LORegion.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

This field resets to 0.

**Accessing the LORC_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, LORC_EL1**

```assembly
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTeN == '1') &&
        HFRGTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return LORC_EL1;
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORC_EL1;
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            return LORC_EL1;
    end

MSR LORC_EL1, <Xt>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.LORC_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    LORC_EL1 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    LORC_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  else
    LORC_EL1 = X[t];
  endif
The LOREA_EL1 characteristics are:

**Purpose**

Holds the physical address of the end of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

**Configuration**

This register is present only when ARMv8.1-LOR is implemented. Otherwise, direct accesses to LOREA_EL1 are UNDEFINED.

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

LOREA_EL1 is a 64-bit register.

**Field descriptions**

The LOREA_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | EA[51:48] | EA[47:16] |
| EA[47:16] | RES0 |

Any of the fields in this register are permitted to be cached in a TLB.

**Bits [63:52]**

Reserved, RES0.

**EA[51:48], bits [51:48]**

When ARMv8.2-LPA is implemented:

- This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**EA[47:16], bits [47:16]**

Bits [47:16] of the end physical address of an LORegion described in the current LORegion descriptor selected by LORC_EL1.DS. Bits[15:0] of this address are defined to be 0xFFFF. For implementations with fewer than 48 bits, the upper bits of this field are RES0.
When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, EA[51:48] form the upper part of the address value. Otherwise, for implementations with fewer than 52 physical address bits, EA[51:48] are RES0.

This field resets to an architecturally UNKNOWN value.

**Bits [15:0]**

Reserved, RES0.

## Accessing the LOREA_EL1

Accesses to this register use the following encodings:

### MRS <Xt>, LOREA_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  else
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HGRTR_EL2.ROREA_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return LOREA_EL1;
elsif PSTATE.EL == EL2 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return LOREA_EL1;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  else
    return LOREA_EL1;
```

### MSR LOREA_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.LOREA_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    LOREA_EL1 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    LOREA_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  else
    LOREA_EL1 = X[t];
  endif
The LORID_EL1 characteristics are:

**Purpose**

Indicates the number of LORegions and LORegion descriptors supported by the PE.

**Configuration**

This register is present only when ARMv8.1-LOR is implemented. Otherwise, direct accesses to LORID_EL1 are undefined.

If no LORegion descriptors are implemented, then the registers LORC_EL1, LORN_EL1, LOREA_EL1, and LORSA_EL1 are res0.

**Attributes**

LORID_EL1 is a 64-bit register.

**Field descriptions**

The LORID_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-24</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>23-16</td>
<td>LD, bits [23:16] Number of LORegion descriptors supported by the PE. This is an 8-bit binary number.</td>
</tr>
<tr>
<td>15-8</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>7-0</td>
<td>LR, bits [7:0] Number of LORegions supported by the PE. This is an 8-bit binary number.</td>
</tr>
</tbody>
</table>

**Note**

If LORID_EL1 indicates that no LORegions are implemented, then LoadLOAcquire and StoreLORelease will behave as LoadAcquire and StoreRelease.

**Accessing the LORID_EL1**

Accesses to this register use the following encodings:
LORID_EL1, LORegionID (EL1)

### MRS <Xt>, LORID_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b11</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGTR_EL2.LORID_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return LORID_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORID_EL1;
        elsif PSTATE.EL == EL3 then
            return LORID_EL1;
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
LORN_EL1, LORegion Number (EL1)

The LORN_EL1 characteristics are:

**Purpose**

Holds the number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

**Configuration**

This register is present only when ARMv8.1-LOR is implemented. Otherwise, direct accesses to LORN_EL1 are **UNDEFINED**.

This register is **RES0** if any of the following apply:

- No LORegion descriptors are supported by the PE.
- **LORC_EL1.DS** points to a LORegion that is not supported by the PE.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

LORN_EL1 is a 64-bit register.

**Field descriptions**

The LORN_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 | RES0 | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num | Num |

Any of the fields in this register are permitted to be cached in a TLB.

**Bits [63:8]**

Reserved, **RES0**.

**Num, bits [7:0]**

Number of the LORegion described in the current LORegion descriptor selected by LORC_EL1.DS.

The maximum number of LORegions supported by the PE is 256. If the maximum number is less than 256, then bits[8:N] are **RES0**, where N is \(\log_2(\text{Number of LORegions supported by the PE})\).

If this field points to a LORegion that is not supported by the PE, then the current LORegion descriptor does not match any LORegion.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the LORN_EL1**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return LORN_EL1;
    end if;
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return LORN_EL1;
    end if;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORN_EL1 = X[t];
    end if;
else
    LORN_EL1 = X[t];
end if;

MSR LORN_EL1, <Xt>
LORSA_EL1, LORegion Start Address (EL1)

The LORSA_EL1 characteristics are:

**Purpose**

Indicates whether the current LORegion descriptor selected by LORC_EL1.DS is enabled, and holds the physical address of the start of the LORegion.

**Configuration**

This register is present only when ARMv8.1-LOR is implemented. Otherwise, direct accesses to LORSA_EL1 are UNDEFINED.

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- LORC_EL1.DS points to a LORegion that is not supported by the PE.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

LORSA_EL1 is a 64-bit register.

**Field descriptions**

The LORSA_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>SA[51:48]</td>
<td>SA[47:16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SA[47:16]</td>
<td>RES0</td>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Any of the fields in this register are permitted to be cached in a TLB.

**Bits [63:52]**

Reserved, RES0.

**SA[51:48], bits [51:48]**

*When ARMv8.2-LPA is implemented:*


This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
**SA[47:16], bits [47:16]**

Bits [47:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by LORC_EL1. Bits[15:0] of this address are defined to be 0x0000. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, SA[51:48] form the upper part of the address value. Otherwise, for implementations with fewer than 52 physical address bits, SA[51:48] are RES0.

This field resets to an architecturally **UNKNOWN** value.

**Bits [15:1]**

Reserved, RES0.

**Valid, bit [0]**

Indicates whether the current LORegion Descriptor is enabled.

<table>
<thead>
<tr>
<th>Valid</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disabled</td>
</tr>
<tr>
<td>0b1</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Accessing the LORSA_EL1**

Accesses to this register use the following encodings:

```c
MRS <Xt>, LORSA_EL1
```

```c
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.LORSA_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORSA_EL1;
        endif
    endif
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORSA_EL1;
        endif
    endif
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        return LORSA_EL1;
    endif
else
    return LORSA_EL1;
end
```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TLOR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.LORSA_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    LORSA_EL1 = X[t];
  end if;
elsif PSTATE.EL == EL2 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TLOR == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    LORSA_EL1 = X[t];
  end if;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    UNDEFINED;
  else
    LORSA_EL1 = X[t];
  end if;
MAIR_EL1, Memory Attribute Indirection Register (EL1)

The MAIR_EL1 characteristics are:

**Purpose**

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL1.

**Configuration**

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PRRR[31:0] when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MAIR0[31:0] when TTBCR.EAE == 1.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register NMRR[31:0] when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register MAIR1[31:0] when TTBCR.EAE == 1.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MAIR_EL1 is a 64-bit register.

**Field descriptions**

The MAIR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Attr7 | Attr6 | Attr5 | Attr4 |
| Attr3 | Attr2 | Attr1 | Attr0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

MAIR_EL1 is permitted to be cached in a TLB.

**Attr<n>**, bits [8n+7:8n], for n = 0 to 7


Attr is encoded as follows:
### Device Memory

- **0b0000dd00**: Device memory. See encoding of 'dd' for the type of Device memory.

### Normal Memory

- **0b0000ddxx**: Normal memory. See encoding of 'dd' for the type of Device memory.
- **0b0000iiii**: Normal memory. See encoding of 'iiii' for the type of Normal Memory.
- **0b11110000**: Tagged Normal Memory. Inner+outer Write-Back Non-transient memory, Inner+outer Read-allocate, Inner+outer Write-Allocate.

### Tagged Normal Memory

- **0bxxxx0000**: Tagged Normal Memory. Inner+outer Write-Back Non-transient memory, Inner+outer Read-allocate, Inner+outer Write-Allocate.

'**dd**' is encoded as follows:

<table>
<thead>
<tr>
<th>dd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Device-nGnRnE memory</td>
</tr>
<tr>
<td>0b01</td>
<td>Device-nGnRE memory</td>
</tr>
<tr>
<td>0b10</td>
<td>Device-nGRE memory</td>
</tr>
<tr>
<td>0b11</td>
<td>Device-GRE memory</td>
</tr>
</tbody>
</table>

'**oooo**' is encoded as follows:

<table>
<thead>
<tr>
<th>oooo</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>See encoding of Attr</td>
</tr>
<tr>
<td>0b00RW</td>
<td>Normal memory, Outer Write-Through Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Write-Through Transient</td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Outer Non-cacheable</td>
</tr>
<tr>
<td>0b01RW</td>
<td>Normal memory, Outer Write-Back Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Write-Back Transient</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Outer Write-Through Non-transient</td>
</tr>
<tr>
<td>0b11RW</td>
<td>Normal memory, Outer Write-Back Non-transient</td>
</tr>
</tbody>
</table>

R = Outer Read-allocate policy, W = Outer Write-allocate policy.

'**iiii**' is encoded as follows:

<table>
<thead>
<tr>
<th>iii</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>See encoding of Attr</td>
</tr>
<tr>
<td>0b00RW</td>
<td>Normal memory, Inner Write-Through Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Write-Through Transient</td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Inner Non-cacheable</td>
</tr>
<tr>
<td>0b01RW</td>
<td>Normal memory, Inner Write-Back Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Write-Back Transient</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Inner Write-Through Non-transient</td>
</tr>
<tr>
<td>0b11RW</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
</tbody>
</table>

R = Inner Read-allocate policy, W = Inner Write-allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

<table>
<thead>
<tr>
<th>R or W</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Allocate</td>
</tr>
<tr>
<td>0b1</td>
<td>Allocate</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### Accessing the MAIR_EL1

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL3 using the mnemonic MAIR_EL1 or MAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, MAIR_EL1

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
             HFGWTR_EL2.MAIR_EL1 == '1' then
    return NVMem[0x140];
  else
    return MAIR_EL1;
  endif 
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return MAIR_EL2;
  else
    return MAIR_EL1;
  endif
elsif PSTATE.EL == EL3 then
  return MAIR_EL1;
```

MSR MAIR_EL1, <Xt>

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
             HFGWTR_EL2.MAIR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return NVMem[0x140] = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    MAIR_EL2 = X[t];
  else
    MAIR_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  MAIR_EL1 = X[t];
```

MRS <Xt>, MAIR_EL12

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

MAIR_EL1, Memory Attribute Indirection Register (EL1)
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x140];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return MAIR_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return MAIR_EL1;
  else
    UNDEFINED;

MSR MAIR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x140] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    MAIR_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    MAIR_EL1 = X[t];
  else
    UNDEFINED;
MAIR_EL2, Memory Attribute Indirection Register (EL2)

The MAIR_EL2 characteristics are:

**Purpose**

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL2.

**Configuration**

AArch64 System register MAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HMAIR0[31:0].

AArch64 System register MAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register HMAIR1[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MAIR_EL2 is a 64-bit register.

**Field descriptions**

The MAIR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Attr7 | Attr6 | Attr5 | Attr4 |
| Attr3 | Attr2 | Attr1 | Attr0 |

MAIR_EL2 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 0 to 7


Attr is encoded as follows:
### Attrib | Meaning
--- | ---
0b0000dd00 | Device memory. See encoding of 'dd' for the type of Device memory.
0b0000ddxx, (xx != 00) | UNPREDICTABLE
0b0000iii, (oooo != 0000 and iii != 0000) | Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal Memory.
0b11110000 | Tagged Normal Memory. Inner+Outer Write-Back Non-Transient memory, Inner+Outer Read-Allocate, Inner+Outer Write-Allocate.
0bxxxx0000, (xxxx != 0000 and xxxx != 1111) | UNPREDICTABLE

'dd' is encoded as follows:

| dd | Meaning |
--- | ---|
0b00 | Device-nGnRnE memory
0b01 | Device-nGnRE memory
0b10 | Device-nGRE memory
0b11 | Device-GRE memory

'oooo' is encoded as follows:

| 'oooo' | Meaning |
--- | ---|
0b0000 | See encoding of Attr
0b00RW, RW not | Normal memory, Outer Write-Through Transient
0b00 | Normal memory, Outer Non-cacheable
0b01RW, RW not | Normal memory, Outer Write-Back Transient
0b00 | Normal memory, Outer Write-Through Non-transient
0b01RW | Normal memory, Outer Write-Back Non-transient

'R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

'iiii' is encoded as follows:

| 'iiii' | Meaning |
--- | ---|
0b0000 | See encoding of Attr
0b00RW, RW not | Normal memory, Inner Write-Through Transient
0b00 | Normal memory, Inner Non-cacheable
0b01RW, RW not | Normal memory, Inner Write-Back Transient
0b00 | Normal memory, Inner Write-Through Non-transient
0b01RW | Normal memory, Inner Write-Back Non-transient

'R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

| R or W | Meaning |
--- | ---|
0b0 | No Allocate
0b1 | Allocate

This field resets to an architecturally UNKNOWN value.

### Accessing the MAIR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic MAIR_EL2 or MAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, MAIR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elifs PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elifs PSTATE.EL == EL2 then
  return MAIR_EL2;
elifs PSTATE.EL == EL3 then
  return MAIR_EL2;

MSR MAIR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elifs PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elifs PSTATE.EL == EL2 then
  MAIR_EL2 = X[t];
elifs PSTATE.EL == EL3 then
  MAIR_EL2 = X[t];

MRS <Xt>, MAIR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elifs PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.MAIR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x140];
  else
    return MAIR_EL1;
elifs PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return MAIR_EL2;
elifs PSTATE.EL == EL3 then
  return MAIR_EL1;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
  HFGWTR_EL2.MAIR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x140] = X[t];
  else
    MAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    MAIR_EL2 = X[t];
else
  MAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  MAIR_EL1 = X[t];
MAIR_EL3, Memory Attribute Indirection Register (EL3)

The MAIR_EL3 characteristics are:

**Purpose**

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to MAIR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MAIR_EL3 is a 64-bit register.

**Field descriptions**

The MAIR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>Attr7</th>
<th>Attr6</th>
<th>Attr5</th>
<th>Attr4</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Attr3</td>
<td>Attr2</td>
<td>Attr1</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Attr0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

MAIR_EL3 is permitted to be cached in a TLB.

**Attr,<n>, bits [8n+7:8n], for n = 0 to 7**


Attr is encoded as follows:

<table>
<thead>
<tr>
<th>Attr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000dd00</td>
<td>Device memory. See encoding of 'dd' for the type of Device memory.</td>
</tr>
<tr>
<td>0b0000ddxx, (xx != 00)</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td>0b0000iiii, (oooo != 0000 and iiii != 0000)</td>
<td>Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal Memory.</td>
</tr>
<tr>
<td>0b11110000</td>
<td>Tagged Normal Memory. Inner+Outer Write-Back Non-Transient memory, Inner+Outer Read-Allocate, Inner+Outer Write-Allocate.</td>
</tr>
<tr>
<td>0bxxxx0000, (xxxx != 0000 and xxxx != 1111)</td>
<td>UNPREDICTABLE</td>
</tr>
</tbody>
</table>

'dd' is encoded as follows:
MAIR_EL3, Memory Attribute Indirection Register (EL3)

---

<table>
<thead>
<tr>
<th>dd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Device-nGnRnE memory</td>
</tr>
<tr>
<td>0b01</td>
<td>Device-nGnRE memory</td>
</tr>
<tr>
<td>0b10</td>
<td>Device-nGRE memory</td>
</tr>
<tr>
<td>0b11</td>
<td>Device-GRE memory</td>
</tr>
</tbody>
</table>

'o000' is encoded as follows:

<table>
<thead>
<tr>
<th>'oooo'</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>See encoding of Attr</td>
</tr>
<tr>
<td>0b00RW, RW not</td>
<td>Normal memory, Outer Write-Through Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable</td>
</tr>
<tr>
<td>0b01RW, RW not</td>
<td>Normal memory, Outer Write-Back Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Write-Through Non-transient</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Outer Write-Back Non-transient</td>
</tr>
</tbody>
</table>

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

'iiii' is encoded as follows:

<table>
<thead>
<tr>
<th>'iiii'</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>See encoding of Attr</td>
</tr>
<tr>
<td>0b00RW, RW not</td>
<td>Normal memory, Inner Write-Through Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable</td>
</tr>
<tr>
<td>0b01RW, RW not</td>
<td>Normal memory, Inner Write-Back Transient</td>
</tr>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Write-Through Non-transient</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
</tbody>
</table>

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

<table>
<thead>
<tr>
<th>R or W</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Allocate</td>
</tr>
<tr>
<td>0b1</td>
<td>Allocate</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the MAIR_EL3**

Accesses to this register use the following encodings:

`MRS <Xt>, MAIR_EL3`

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MAIR_EL3;
**MSR MAIR_EL3, `<Xt>`**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  MAIR_EL3 = X[t];
```
MDCCINT_EL1, Monitor DCC Interrupt Enable Register

The MDCCINT_EL1 characteristics are:

**Purpose**

Enables interrupt requests to be signaled based on the DCC status flags.

**Configuration**

AArch64 System register MDCCINT_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDCCINT[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MDCCINT_EL1 is a 64-bit register.

**Field descriptions**

The MDCCINT_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit (63:31)</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59 58 57 56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55 54 53 52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51 50 49 48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47 46 45 44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43 42 41 40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39 38 37 36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35 34 33 32</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:31]**

Reserved, RES0.

**RX, bit [30]**

DCC interrupt request enable control for DTRRX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

<table>
<thead>
<tr>
<th>RX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt request generated by DTRRX.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt request will be generated on RXfull == 1.</td>
</tr>
</tbody>
</table>

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

On a Warm reset, this field resets to 0.

**TX, bit [29]**

DCC interrupt request enable control for DTRTX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

<table>
<thead>
<tr>
<th>TX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt request generated by DTRTX.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt request will be generated on TXfull == 0.</td>
</tr>
</tbody>
</table>

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

On a Warm reset, this field resets to 0.
MDCCINT_EL1, Monitor DCC Interrupt Enable Register

**Bits [28:0]**

Reserved, RES0.

**Accessing the MDCCINT_EL1**

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ MDCCINT_EL1}
\]

<table>
<thead>
<tr>
<th>(\text{op0})</th>
<th>(\text{op1})</th>
<th>(\text{CRn})</th>
<th>(\text{CRm})</th>
<th>(\text{op2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MDCCINT_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return MDCCINT_EL1;
  elsif PSTATE.EL == EL3 then
    return MDCCINT_EL1;

\[
\text{MSR } \text{MDCCINT_EL1}, <Xt>
\]

<table>
<thead>
<tr>
<th>(\text{op0})</th>
<th>(\text{op1})</th>
<th>(\text{CRn})</th>
<th>(\text{CRm})</th>
<th>(\text{op2})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t];
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MDCCINT_EL1 = X[t];
MDCCSR_EL0, Monitor DCC Status Register

The MDCCSR_EL0 characteristics are:

**Purpose**

Read-only register containing control status flags for the DCC.

**Configuration**

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to External register EDSCR[30:29].

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to AArch32 System register DBGDSCRint[30:29].

**Attributes**

MDCCSR_EL0 is a 64-bit register.

**Field descriptions**

The MDCCSR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RXfull | TXfull | RES0 | RAZ | RES0 | RAZ | RES0 | RAZ | RES0 | RAZ | RES0 |

- **Bits [63:31]**
  - Reserved, RES0.

- **RXfull, bit [30]**
  - DTRRX full. Read-only view of the equivalent bit in the EDSCR.

- **TXfull, bit [29]**
  - DTRTX full. Read-only view of the equivalent bit in the EDSCR.

- **Bits [28:19]**
  - Reserved, RES0.

- **Bits [18:15]**
  - Reserved, RAZ.

- **Bits [14:13]**
  - Reserved, RES0.
Bit [12]
Reserved, RAZ.

Bits [11:6]
Reserved, RES0.

Bits [5:2]
Reserved, RAZ.

Bits [1:0]
Reserved, RES0.

Accessing the MDCCSR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, MDCCSR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MDCCSR_EL0;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return MDCCSR_EL0;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return MDCCSR_EL0;
  elsif PSTATE.EL == EL3 then
    return MDCCSR_EL0;
The MDCR_EL2 characteristics are:

**Purpose**

Provides EL2 configuration options for self-hosted debug and the Performance Monitors Extension.

**Configuration**

AArch64 System register MDCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register \( \text{HDCR}[31:0] \).

If EL2 is not implemented, this register is \( \text{RES0} \) from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally \( \text{UNKNOWN} \) values.

**Attributes**

MDCR_EL2 is a 64-bit register.

**Field descriptions**

The MDCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:29</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**MTPME, bit [28]**

When \( \text{ARMv8.6-MTPMU} \) is implemented and \( \texttt{!HaveEL(EL3)} \):

Multi-threaded PMU Enable. Enables use of the \( \text{PMEVTYPE}<n>_{\text{EL0}}.\text{MT} \) bits.

<table>
<thead>
<tr>
<th>MTPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ARMv8.6-MTPMU is disabled. The Effective value of ( \text{PMEVTYPE}&lt;n&gt;_{\text{EL0}}.\text{MT} ) is zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>( \text{PMEVTYPE}&lt;n&gt;_{\text{EL0}}.\text{MT} ) bits not affected by this bit.</td>
</tr>
</tbody>
</table>

If ARMv8.6-MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is \( \text{IMPLEMENTATION DEFINED} \) whether the PE behaves as if this bit is 0.

On a Cold reset, this field resets to 1.

**Otherwise:**

Reserved, RES0.
**TDCC, bit [27]**

When ARMv8.6-FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

<table>
<thead>
<tr>
<th>TDCC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any register accesses to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Trap exception to EL2, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.</td>
</tr>
</tbody>
</table>

The DCC registers trapped by this control are:

AArch64: **OSDTRRX_EL1**, **OSDTRTX_EL1**, **MDCCSR_EL0**, **MDCCINT_EL1**, and, when the PE is in Non-debug state, **DBGDTR_EL0**, **DBGDTRRX_EL0**, and **DBGDTRTX_EL0**.

AArch32: **DBGDTRRXtext**, **DBGDTRTXtext**, **DBGDSRInt**, **DBGDCINT**, and, when the PE is in Non-debug state, **DBGDTRRXint** and **DBGDTRTXint**.

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to **DBGDTRTXint** and STC from **DBGDTRRXint**.
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL2.TDCC does not trap any accesses to:

AArch64: **DBGDTR_EL0**, **DBGDTRRX_EL0**, and **DBGDTRTX_EL0**.

AArch32: **DBGDTRRXint** and **DBGDTRTXint**.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HLP, bit [26]**

When ARMv8.5-PMU is implemented:

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

<table>
<thead>
<tr>
<th>HLP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counter overflow on increment that causes unsigned overflow of <strong>PMEVCNTR&lt;n&gt;_EL0[31:0]</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counter overflow on increment that causes unsigned overflow of <strong>PMEVCNTR&lt;n&gt;_EL0[63:0]</strong>.</td>
</tr>
</tbody>
</table>

If MDCR_EL2.HPMN is less than **PMCR_EL0.N** or **PMCR.N**, this bit affects the operation of event counters in the range [MDCR_EL2.HPMN..(PMCR_EL0.N-1)] or [MDCR_EL2.HPMN..(PMCR.N-1)]. Otherwise this bit has no effect on the operation of the event counters.

**Note**

The effect of MDCR_EL2.HPMN on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, and regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the MDCR_EL2.HPMN field.
On a Warm reset, this field resets to an architecturally `UNKNOWN` value.

Otherwise:

Reserved, RES0.

**E2TB, bits [25:24]**

When TRBE is implemented:

EL2 Trace Buffer. If EL2 is implemented and enabled in the current Security state, controls the owning translation regime and access to Trace Buffer control registers from EL1.

<table>
<thead>
<tr>
<th>E2TB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>If EL2 is implemented and enabled in the current Security state, Trace Buffer owning Exception level is EL2. Accesses to Trace Buffer controls at EL1 generate a Trap exception to EL2.</td>
</tr>
<tr>
<td>0b10</td>
<td>Trace Buffer owning Exception level is EL1. If TraceBufferEnabled() == TRUE, tracing is prohibited at EL2. If EL2 is implemented and enabled in the current Security state, accesses to Trace Buffer controls at EL1 generate a Trap exception to EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>Trace Buffer owning Exception level is EL1. If TraceBufferEnabled() == TRUE, tracing is prohibited at EL2.</td>
</tr>
</tbody>
</table>

The Trace Buffer control registers trapped by this control are: TRBBASER_EL1, TRBLIMITR_EL1, TRBMAR_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.

If EL2 is not implemented or is disabled, the PE behaves as if this field is 0b11, other than for a direct read of the register.

On a Warm reset, this field resets to an architecturally `UNKNOWN` value.

Otherwise:

Reserved, RES0.

**HCCD, bit [23]**

When ARMv8.5-PMU is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting at EL2.

<table>
<thead>
<tr>
<th>HCCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting by PMCCNTR_EL0 is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting by PMCCNTR_EL0 is prohibited at EL2.</td>
</tr>
</tbody>
</table>

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

**Bits [22:20]**

Reserved, RES0.
**TTRF, bit [19]**

When ARMv8.4-Trace is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2, as follows:

- Access to TRFCR_EL1 is trapped to EL2, reported using EC syndrome value 0x18.
- Access to TRFCR is trapped to EL2, reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>TTRF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to TRFCR_EL1 and TRFCR at EL1 are not affected by this control.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to TRFCR_EL1 and TRFCR at EL1 generate a trap exception to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**Bit [18]**

Reserved, RES0.

**HPMD, bit [17]**

When ARMv8.1-PMU is implemented:

Guest Performance Monitors Disable. This control prohibits event counting at EL2.

<table>
<thead>
<tr>
<th>HPMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counting allowed at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counting prohibited at EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-Debug is not implemented, event counting is prohibited unless enabled by the IMPLEMENTATION DEFINED authentication interface `ExternalSecureNoninvasiveDebugEnabled()`.

This control applies only to:

- The event counters in the range [0..(MDCR_EL2.HPMN-1)].
- If PMCR_EL0.DP is set to 1, PMCCNTR_EL0.

The other event counters are unaffected, and when PMCR_EL0.DP is set to 0, PMCCNTR_EL0 is unaffected.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

**Bits [16:15]**

Reserved, RES0.

**TPMS, bit [14]**

When SPE is implemented:

Trap Performance Monitor Sampling. When EL2 is enabled in the current Security state, this field controls access to Statistical Profiling control registers from EL1.
<table>
<thead>
<tr>
<th>TPMS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not trap Statistical Profiling controls to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to Statistical Profiling controls at EL1 generate a Trap exception to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

The Statistical Profiling control registers trapped by this control are: `PMSCR_EL1`, `PMSEVFR_EL1`, `PMSFCR_EL1`, `PMSICR_EL1`, `PMSIDR_EL1`, `PMSIRR_EL1`, and `PMSLATFR_EL1`.

Otherwise:

Reserved, RES0.

**E2PB, bits [13:12]**

When SPE is implemented:

EL2 Profiling Buffer. If EL2 is implemented and enabled in the Profiling Buffer owning Security state, this field controls the owning translation regime. If EL2 is implemented and enabled in the current Security state, this field controls access to Profiling Buffer control registers from EL1.

<table>
<thead>
<tr>
<th>E2PB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>If EL2 is implemented and enabled in the Profiling Buffer owning Security state, the Profiling Buffer uses the EL2 or EL2&amp;0 stage 1 translation regime. Otherwise the Profiling Buffer uses the EL1&amp;0 stage 1 translation regime. If EL2 is implemented and enabled in the current Security state, accesses to Profiling Buffer control registers at EL1 generate a Trap exception to EL2.</td>
</tr>
<tr>
<td>0b10</td>
<td>Profiling Buffer uses the EL1&amp;0 stage 1 translation regime. If EL2 is implemented and enabled in the current Security state, accesses to Profiling Buffer control registers at EL1 generate a Trap exception to EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>Profiling Buffer uses the EL1&amp;0 stage 1 translation regime. Accesses to Profiling Buffer control registers at EL1 are not trapped to EL2.</td>
</tr>
</tbody>
</table>

All other values are reserved.

The Profiling Buffer control registers trapped by this control are: `PMBLIMITR_EL1`, `PMBPTR_EL1`, and `PMBSR_EL1`.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**TDRA, bit [11]**

Trap Debug ROM Address register access. Traps System register accesses to the Debug ROM registers to EL2 when EL2 is enabled in the current Security state as follows:

- If EL1 is using AArch64 state, accesses to `MDRAR_EL1` are trapped to EL2, reported using EC syndrome value 0x18.
- If EL0 or EL1 is using AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05 and MRRC or MCRR accesses are trapped to EL2, reported using EC syndrome value 0x0C:
  - `DBGDRAR`, `DBGDSAR`.

<table>
<thead>
<tr>
<th>TDRA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 and EL1 System register accesses to the Debug ROM registers are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by <code>DBGDSCRext</code>, UDCCdis or <code>MDSCR_EL1</code>.TDCC.</td>
</tr>
</tbody>
</table>
This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- `MDCR_EL2.TDE == 1`.
- `HCR_EL2.TGE == 1`.

## Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

On a Warm reset, this field resets to an architecturally `UNKNOWN` value.

### TDOSA, bit [10]

**When ARMv8.0-DoubleLock is implemented:**

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x18`:
  - `OSLAR_EL1`, `OSLSR_EL1`, `OSDLR_EL1`, and `DBGPRCR_EL1`.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x05`:
  - `DBGOSLSR`, `DBGOSLAR`, `DBGOSDLR`, and `DBGPRCR`.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

<table>
<thead>
<tr>
<th>TDOSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

## Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- `MDCR_EL2.TDE == 1`.
- `HCR_EL2.TGE == 1`.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

On a Warm reset, this field resets to an architecturally `UNKNOWN` value.

**Otherwise:**

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value `0x18`:
  - `OSLAR_EL1`, `OSLSR_EL1`, and `DBGPRCR_EL1`.
  - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05:

- DBGOSLSR, DBGOSLAR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 are trapped and DBGOSDLR are trapped.

<table>
<thead>
<tr>
<th>TDOSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

**Note**

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- MDCR_EL2.TDE == 1.
- HCR_EL2.TGE == 1.

**Note**

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**TDA, bit [9]**

Trap Debug Access. Traps EL0 and EL1 System register accesses to debug System registers that are not trapped by MDCR_EL2.TDRA or MDCR_EL2.TDOSA, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 reported using EC syndrome value 0x18:
  - MDCCSR_EL0, MDCCINT_EL1, OSDTRRX_EL1, MDSCR_EL1, OSDTRTX_EL1, OSECER_EL1, DBGVR<n>_EL1, DBGBCR<n>_EL1, DBGWVR<n>_EL1, DBGWCR<n>_EL1, DBGCLAIMSET_EL1, DBGCLAIMCLR_EL1, DBGAUTHSTATUS_EL1.
  - When not in Debug state, DBGDTR_EL0, DBGDTRRX_EL0, DBGDTRTX_EL0.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05:
  - DBGDIR, DBGDCRint, DBGDCINT, DBGWAFAR, DBGVCRC, DBGDCRext, DBGDTRXext, DBGVR<n>, DBGBCR<n>, DBGBXVR<n>, DBGWCR<n>, DBGWVR<n>, DBGCLAIMSET, DBGCLAIMCLR, DBGAUTHSTATUS, DBGDEVID, DBGDEVID1, DBGDEVID2, DBGOSERRCR.
  - When not in Debug state, DBGDTRRXint and DBGDTRTXint.
- In AArch32 state, STC accesses to DBGDTRRXint and LDC accesses to DBGDTRTXint are trapped to EL2, reported using EC syndrome value 0x06.

<table>
<thead>
<tr>
<th>TDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 or EL1 System register accesses to the debug registers are trapped from both Execution states to EL2 when EL2 is enabled in the current Security state, unless the access generates a higher priority exception.</td>
</tr>
</tbody>
</table>

Traps of AArch32 accesses to DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

Traps of AArch64 accesses to DBGDT_ELO, DBGDTRRX_ELO, and DBGDTRTX_ELO are ignored in Debug state.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:
• **MDCR_EL2.TDE == 1**
• **HCR_EL2.TGE == 1**

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

### TDE, bit [8]

**Trap Debug exceptions.**

<table>
<thead>
<tr>
<th>TDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the routing of debug exceptions, and has no effect on accesses to debug registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>Debug exceptions generated at EL1 or EL0 are routed to EL2 when EL2 is enabled in the current Security state. The MDCR_EL2.(TDRA, TDOSA, TDA) fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.</td>
</tr>
</tbody>
</table>

This field is treated as being 1 for all purposes other than a direct read when **HCR_EL2.TGE == 1**.

### HPME, bit [7]

When PMUv3 is implemented:

**[MDCR_EL2.HPMN..(N-1)] event counters enable.**

<table>
<thead>
<tr>
<th>HPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counters in the range [MDCR_EL2.HPMN..(PMCR_EL0.N-1)] are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counters in the range [MDCR_EL2.HPMN..(PMCR_EL0.N-1)] are enabled by PMCNTENSET_EL0.</td>
</tr>
</tbody>
</table>

If MDCR_EL2.HPMN is less than PMCR_EL0.N or PMCR.N, the event counters in the range [MDCR_EL2.HPMN..(PMCR_EL0.N-1)] or [HDCR.HPMN..(PMCR.N-1)], are enabled and disabled by this bit. Otherwise this bit has no effect on the operation of the event counters.

**Note**

The effect of MDCR_EL2.HPMN on the operation of this bit applies regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the HPMN field.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

### Otherwise:

Reserved, RES0.

### TPM, bit [6]

When PMUv3 is implemented:

**Trap Performance Monitors accesses.** Traps EL0 and EL1 accesses to all Performance Monitor registers to EL2 when EL2 is enabled in the current Security state, from both Execution states, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
  - **PMCR_EL0**, **PMCNTENSET_EL0**, **PMCNTENCLR_EL0**, **PMOVSCLR_EL0**, **PMSWINC_EL0**, **PMSERL_EL0**, **PMCEID0_EL0**, **PMCEID1_EL0**, **PMCCNTR_EL0**, **PMXEVTYPEP_EL0**.
PMXEVCNTR_EL0, PMUSERENR_EL0, PMINTENSET_EL1, PMINTENCLR_EL1, PMOVSSET_EL0, PMEVCNTR<n>_EL0, PMEVTYPER<n>_EL0, PMEZEVNTR_EL0, PMCCFILTR_EL0.

- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, MRRC or MCRR accesses are trapped to EL2 and reported using EC syndrome value 0x04:
  - PMCR, PMCNTENSET, PMCNTENCLR, PMOVSR, PMSWINC, PMSELR, PMCEID0, PMCEID1, PMCCNTR, PMXEVTPER, PMXENVTR, PMUSERENR, PMINTENSET, PMINTENCLR, PMOVSSET, PMEVCNTR<n>, PMEVTYPER<n>, PMCCFILTR.

- If ARMv8.4-PMU is implemented, in AArch64 state, PMMIR_EL1, and in AArch32 state, PMMIR.

- If ARMv8.1-PMU is implemented, in AArch32 state, PMCEID2, and PMCEID3.

<table>
<thead>
<tr>
<th>TPM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 and EL1 accesses to all Performance Monitor registers are trapped to EL2 when EL2 is enabled in the current Security state.</td>
</tr>
</tbody>
</table>

**Note**

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**TPMCR, bit [5]**

*When PMUv3 is implemented:*

Trap PMCR_EL0 or PMCR accesses. Traps EL0 and EL1 accesses to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to PMCR_EL0 are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, accesses to PMCR are trapped to EL2, reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>TPMCR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 and EL1 accesses to the PMCR_EL0 or PMCR are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by PMUSERENR.EN or PMUSERENR_EL0.EN.</td>
</tr>
</tbody>
</table>

**Note**

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
HPMN, bits [4:0]

When PMUv3 is implemented:

Defines the number of event counters that are accessible from EL3, EL2, EL1, and from EL0 if permitted.

If HPMN is less than \( \text{PMCR\_EL0}\_N \), HPMN divides the Performance Monitors into two ranges: \( 0..(\text{HPMN}-1) \) and \( \text{HPMN..(PMCR\_EL0\_N)-1} \).

For an event counter in the range \( 0..(\text{HPMN}-1) \):

- The counter is accessible from EL3, EL2, and EL1, and from EL0 if permitted by \( \text{PMUSERENR\_EL0} \) or \( \text{PMUSERENR} \).
- If Armv8.5-PMU is implemented, \( \text{PMCR\_EL0}\_LP \) or \( \text{PMCR\_LP} \) determines whether the counter overflow flag is set on unsigned overflow of \( \text{PMEVCNTR}<n>\_\text{EL0}[31:0] \) or \( \text{PMEVCNTR}<n>\_\text{EL0}[63:0] \).
- The counter is enabled by \( \text{PMCR\_EL0}\_E \) or \( \text{PMCR\_E} \) and bit \( <n> \) of \( \text{PMCNTENSET\_EL0} \).

**Note**

If HPMN is equal to \( \text{PMCR\_EL0}\_N \), this applies to all event counters.

If HPMN is less than \( \text{PMCR\_EL0}\_N \), for an event counter in the range \( \text{HPMN..(PMCR\_EL0\_N)-1} \):

- The counter is accessible from EL2 and EL3.
- If Armv8.4-SecEL2 is disabled or is not implemented, the counter is also accessible from Secure EL1, and from Secure EL0 if permitted by \( \text{PMUSERENR\_EL0} \).
- If Armv8.5-PMU is implemented, MDCR\_EL2.HLP or \( \text{HDCR\_HLP} \) determines whether the counter overflow flag is set on unsigned overflow of \( \text{PMEVCNTR}<n>\_\text{EL0}[31:0] \) or \( \text{PMEVCNTR}<n>\_\text{EL0}[63:0] \).
- The counter is enabled by MDCR\_EL2.HPME or \( \text{HDCR\_HPME} \) and bit \( <n> \) of \( \text{PMCNTENSET\_EL0} \).

If this field is set to 0, or to a value larger than \( \text{PMCR\_EL0}\_N \), then the following \text{CONSTRAINED UNPREDICTABLE} behaviors apply:

- The value returned by a direct read of MDCR\_EL2.HPMN is \text{UNKNOWN}.
- Either:
  - An \text{UNKNOWN} number of counters are reserved for EL2 and EL3 use. That is, the PE behaves as if MDCR\_EL2.HPMN is set to an \text{UNKNOWN} non-zero value less than or equal to \( \text{PMCR\_EL0}\_N \).
  - All counters are reserved for EL2 and EL3 use, meaning no counters are accessible from EL1 and EL0.

On a Warm reset, this field resets to the value in \( \text{PMCR\_EL0}\_N \).

**Otherwise:**

Reserved, RES0.

### Accessing the MDCR\_EL2

Accesses to this register use the following encodings:

\[
\begin{array}{|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b11 & 0b100 & 0b0001 & 0b0001 & 0b001 \\
\hline
\end{array}
\]

MRS <Xt>, MDCR\_EL2
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCR_EL2;
elsif PSTATE.EL == EL3 then
    return MDCR_EL2;

MSR MDCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MDCR_EL2 = X[t];
The MDCR_EL3 characteristics are:

**Purpose**

Provides EL3 configuration options for self-hosted debug and the Performance Monitors Extension.

**Configuration**

AArch64 System register MDCR_EL3 bits [31:0] can be mapped to AArch32 System register SDCR[31:0], but this is not architecturally mandated.

This register is present only when HaveEL(EL3). Otherwise, direct accesses to MDCR_EL3 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MDCR_EL3 is a 64-bit register.

**Field descriptions**

The MDCR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:29]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MTPME, bit [28]</td>
<td>When ARMv8.6-MTPMU is implemented:</td>
</tr>
<tr>
<td>Multi-threaded PMU Enable. Enables use of the PMEVTYPER&lt;n&gt;_EL0.MT bits.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MTPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ARMv8.6-MTPMU is disabled. The Effective value of PMEVTYPER&lt;n&gt;_EL0.MT is zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>PMEVTYPER&lt;n&gt;_EL0.MT bits not affected by this bit.</td>
</tr>
</tbody>
</table>

If ARMv8.6-MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this bit is 0.

On a Cold reset, this field resets to 1.

**Otherwise:**

Reserved, RES0.
TDCC, bit [27]

When ARMv8.6-FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL2, EL1, and EL0 to EL3.

<table>
<thead>
<tr>
<th>TDCC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any register accesses to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to the DCC registers at EL2, EL1, and EL0 generate a Trap exception to EL3, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.</td>
</tr>
</tbody>
</table>

The DCC registers trapped by this control are:

AArch64: **OSDTRRX EL1, OSDTRTX EL1, MDCCSR EL0, MDCCINT EL1**, and, when the PE is in Non-debug state, **DBGDTR EL0, DBGDTRRX EL0, and DBGDTRTX EL0**.

AArch32: **DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCCINT**, and, when the PE is in Non-debug state, **DBGDTRRXint and DBGDTRTXint**.

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to **DBGDTRTXint** and STC from **DBGDTRRXint**.
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL3.TDCC does not trap any accesses to:

AArch64: **DBGDTR EL0, DBGDTRRX EL0**, and **DBGDTRTX EL0**.

AArch32: **DBGDTRRXint** and **DBGDTRTXint**.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

NSTB, bits [25:24]

When TRBE is implemented:

Non-secure Trace Buffer. Controls the owning translation regime and accesses to Trace Buffer control registers from EL2 and EL1.
Trace Buffer owning security state is Secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3.

Trace Buffer owning security state is Secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure state. Accesses to Trace Buffer control registers at EL2 and EL1 in Non-secure state generate Trap exceptions to EL3.

Trace Buffer owning security state is Non-secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Secure state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3.

Trace Buffer owning security state is Non-secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Secure state. Accesses to Trace Buffer control registers at EL2 and EL1 in Secure state generate Trap exceptions to EL3.

The Trace Buffer control registers trapped by this control are: TRBBASER_EL1, TRBLIMITR_EL1, TRBMAP_EL1, TRBPTR_EL1, TRBSR_EL1, and TRBTRG_EL1.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0b1, the PE behaves as if this field is 0b11. If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0b0, the PE behaves as if this field is 0b01.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCCD, bit [23]

When ARMv8.5-PMU is implemented:

Secure Cycle Counter Disable. Prohibits PMCCNTR_EL0 from counting in Secure state.

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

ETAD, bit [22]

When TRBE is implemented:

External Trace Non-secure Access Disable. Controls Non-secure access to PE Trace Unit registers by an external debugger.

If the Trace Extension does not support external debug interface accesses this bit is RES0.

Otherwise, if EL3 is not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b1.
On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

**EPMAD, bit [21]**

*When ARMv8.4-Debug is implemented and PMUv3 is implemented:*

External Performance Monitors Non-secure Access Disable. Controls Non-secure access to Performance Monitor registers by an external debugger.

<table>
<thead>
<tr>
<th>EPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure access to Performance Monitor registers from external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure access to Performance Monitor registers from external debugger is not permitted.</td>
</tr>
</tbody>
</table>

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.

Otherwise, if EL3 is not implemented and the Effective value of SCR_EL3, NS is 0b0, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

*When PMUv3 is implemented:*

External Performance Monitors Access Disable. Controls access to Performance Monitor registers by an external debugger.

<table>
<thead>
<tr>
<th>EPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to Performance Monitor registers from external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to Performance Monitor registers from external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface.</td>
</tr>
</tbody>
</table>

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.

Otherwise, if EL3 is not implemented and the Effective value of SCR_EL3, NS is 0b0, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

**EDAD, bit [20]**

*When ARMv8.4-Debug is implemented:*

External Debug Non-secure Access Disable. Controls Non-secure access to breakpoint, watchpoint, and OSLAR_EL1 registers by an external debugger.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure access to debug registers from external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure access to breakpoint and watchpoint registers, and OSLAR_EL1 from external debugger is not permitted.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR_EL3, NS is 0b0, then the Effective value of this field is 0b1.
On a Warm reset, this field resets to 0.

When ARMv8.2-Debug is implemented:

External Debug Access Disable. Controls access to breakpoint, watchpoint, and OSLAR_EL1 registers by an external debugger.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to debug registers, and to OSLAR_EL1 from external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to breakpoint and watchpoint registers, and to OSLAR_EL1 from external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b1.

On a Warm reset, this field resets to 0.

Otherwise:

External Debug Access disable. Controls access to breakpoint, watchpoint, and optionally OSLAR_EL1 registers by an external debugger.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to debug registers from external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to breakpoint and watchpoint registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. It is IMPLEMENTATION DEFINED whether access to the OSLAR_EL1 register from an external debugger is permitted or not permitted.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b1.

On a Warm reset, this field resets to 0.

TTRF, bit [19]

When ARMv8.4-Trace is implemented:

Trap Trace Filter controls. Traps use of the Trace Filter control registers at EL2 and EL1 to EL3.

The Trace Filter registers trapped by this control are:

- TRFCR_EL2, TRFCR_EL12, TRFCR_EL1, reported using EC syndrome value 0x18.
- HTRFCR and TRFCR, reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>TTRF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to Trace Filter registers at EL2 and EL1 are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to Trace Filter registers at EL2 and EL1 generate a Trap exception to EL3, unless the access generates a higher priority exception.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.
**STE, bit [18]**

**When ARMv8.4-Trace is implemented:**

Secure Trace enable. Enables tracing in Secure state.

<table>
<thead>
<tr>
<th>STE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace prohibited in Secure state unless overridden by the [IMPLEMENTATION DEFINED] authentication interface.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace in Secure state is not affected by this bit.</td>
</tr>
</tbody>
</table>

This bit also controls the level of authentication required by an external debugger to enable external tracing. See 'Register controls to enable self-hosted trace' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL3 is not implemented the Effective value of `SCR_EL3.NS` is 0b0, the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**SPME, bit [17]**

**When ARMv8.2-Debug is implemented and PMUv3 is implemented:**

Secure Performance Monitors enable. This allows event counting in Secure state.

<table>
<thead>
<tr>
<th>SPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counting prohibited in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counting in Secure state not affected by this bit.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented the Effective value of `SCR_EL3.NS` is 0b0, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

**When PMUv3 is implemented:**

Secure Performance Monitors enable. This allows event counting in Secure state.

<table>
<thead>
<tr>
<th>SPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counting prohibited in Secure state, unless <code>ExternalSecureNoninvasiveDebugEnabled()</code> is TRUE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counting in Secure state not affected by this bit.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented the Effective value of `SCR_EL3.NS` is 0b0, then the Effective value of this bit is 0b1.

On a Warm reset, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**SDD, bit [16]**

AArch64 Secure Self-hosted invasive debug disable. Disables Software debug exceptions in Secure state, other than Breakpoint Instruction exceptions.

<table>
<thead>
<tr>
<th>SDD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Debug exceptions in Secure state are not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Debug exceptions, other than Breakpoint Instruction exceptions, are disabled from all Exception levels in Secure state.</td>
</tr>
</tbody>
</table>
The SDD bit is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of SCR_EL3.RW is 0b1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**SPD32, bits [15:14]**

AArch32 Secure self-hosted privileged debug. Enables or disables debug exceptions from Secure EL1 using AArch32, other than Breakpoint Instruction exceptions.

<table>
<thead>
<tr>
<th>SPD32</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Legacy mode. Debug exceptions from Secure EL1 are enabled by the IMPLEMENTATION DEFINED authentication interface.</td>
</tr>
<tr>
<td>0b10</td>
<td>Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled.</td>
</tr>
</tbody>
</table>

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is:

- Ignored if the PE is either:
  - In Non-secure state.
  - In Secure state and Secure EL1 is using AArch64.
- RES0 if the implementation does not support EL1 using AArch32.

If Secure EL1 is using AArch32 then:

- If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER32_EL3.SUIDEN is 0b1.

If EL3 is not implemented and the Effective value of SCR_EL3.NS is 0b0, then the Effective value of this field is 0b11.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSPB, bits [13:12]**

When SPE is implemented:

Non-secure Profiling Buffer. This field controls the owning translation regime and accesses to Statistical Profiling and Profiling Buffer control registers.
NSPB

<table>
<thead>
<tr>
<th>Meaning</th>
<th>NSPB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in both security states generate Trap exceptions to EL3.</td>
<td>0b00</td>
</tr>
<tr>
<td>Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Non-secure state generate Trap exceptions to EL3.</td>
<td>0b01</td>
</tr>
<tr>
<td>Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in both security states generate Trap exceptions to EL3.</td>
<td>0b10</td>
</tr>
<tr>
<td>Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Secure state generate Trap exceptions to EL3.</td>
<td>0b11</td>
</tr>
</tbody>
</table>

The Statistical Profiling and Profiling Buffer control registers trapped by this control are: **PMBLIMITR_EL1**, **PMBPTR_EL1**, **PMBSR_EL1**, **PMSRCR_EL1**, **PMEVFR_EL1**, **PMSFCR_EL1**, **PMSICR_EL1**, **PMSIDR_EL1**, **PMSIRR_EL1**, and **PMSLATR_EL1**.

If EL3 is not implemented and the Effective value of **SCR_EL3.NS** is 0b1, the Effective value of this field is 0b11. If EL3 is not implemented and the Effective value of **SCR_EL3.NS** is 0b0, the Effective value of this field is 0b01.

On a Warm reset, this field resets to an **UNKNOWN** value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES0.

TDOSA, bit [10]

When ARMv8.0-DoubleLock is implemented:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

Accesses to the registers are trapped as follows:

- Accesses from AArch64 state, **OSLR_EL1**, **OSLRSR_EL1**, **OSDLR_EL1**, **DBGPRCR_EL1** and any **IMPLEMENTATION DEFINED** register with similar functionality that the implementation specifies as trapped by this bit, are trapped to EL3 and reported using EC syndrome value 0x18.
- Accesses using MCR or MRC to **DBGOSLAR**, **DBGOSLRSR**, **DBGOSDLR**, and **DBGPRCR**, are trapped to EL3 and reported using EC syndrome value 0x05.
- Accesses to any **IMPLEMENTATION DEFINED** register with similar functionality that the implementation specifies as trapped by this bit.

<table>
<thead>
<tr>
<th>TDOSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by <strong>HDCR.TDOSA</strong> or <strong>MDCR_EL2.TDOSA</strong>.</td>
</tr>
</tbody>
</table>

Note

The powerdown debug registers are not accessible at EL0.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
Otherwise:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

The following registers are affected by this trap:

- **AArch64**: OSLAR_EL1, OSLSR_EL1, and DBGPRCR_EL1.
- **AArch32**: DBGOSLAR, DBGOSLSR, and DBGPRCR.
- **AArch64 and AArch32**: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- It is IMPLEMENTATION DEFINED whether accesses to OSDLR_EL1 and DBGOSDLR are trapped.

<table>
<thead>
<tr>
<th>TDOSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA.</td>
</tr>
</tbody>
</table>

Note

The powerdown debug registers are not accessible at EL0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**TDA, bit [9]**

Trap Debug Access. Traps EL2, EL1, and EL0 System register accesses to those debug System registers that cannot be trapped using the MDCR_EL3.TDOSA field.

Accesses to the debug registers are trapped as follows:

- In AArch64 state, the following registers are trapped to EL3 and reported using EC syndrome value 0x18:
  - DBGBRV<n>_EL1, DBGBCR<n>_EL1, DBGWVR<n>_EL1, DBGWCR<n>_EL1, DBGCLAIMSET_EL1, DBGCLAIMCLR_EL1, DBGAUTHSTATUS_EL1, DBGVR<n>_EL1.
  - AArch64: MDCR_EL2, MDRAR_EL1, MDCCSR_EL0, MDCCINT_EL1, MDSCR_EL1, OSDTRRX_EL1, OSDTRTX_EL1.
- In AArch32 state, SDER is trapped to EL3 and reported using EC syndrome value 0x03.
- In AArch32 state, accesses using MCR or MRC to the following registers are reported using EC syndrome value 0x05, accesses using MCRR or MRRC are reported using EC syndrome value 0x0C:
  - HDCR, DBGDRAR, DBGDSAR, DBGDRAR, DBGDCCINT, DBGWPAR, DBGVCR, DBGVR<n>, DBGBCR<n>, DBGWCR<n>, DBGWVR<n>.
  - DBGCLAIMSET, DBGCLAIMCLR, DBGAUTHSTATUS, DBGDEVID, DBGDEVID1, DBGDEVID2, DBGSECCR.
- In AArch32 state, STC accesses to DBGDTRRXint and LDC accesses to DBGDTRTXint are reported using EC syndrome value 0x06.
- When not in Debug state, the following registers are also trapped to EL3:
  - AArch64 accesses to DBGDTR_EL0, DBGDTRRX_EL0, and DBGDTRTX_EL0, reported using EC syndrome value 0x18.
  - AArch32 accesses using MCR or MRC to DBGDTRXint and DBGDTRTXint, reported using EC syndrome value 0x05.

<table>
<thead>
<tr>
<th>TDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0, EL1, and EL2 accesses to the debug registers, other than the registers that can be trapped by MDCR_EL3.TDOSA, are trapped to EL3, from both Security states and both Execution states, unless it is trapped by DBGDSCRx, UDCCdies, MDSCR_EL1.TDCC, HDCR.TDA or MDCR_EL2.TDOSA.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bits [8:7]**

Reserved, RES0.

**TPM, bit [6]**
When PMUv3 is implemented:

Trap Performance Monitor register accesses. Accesses to all Performance Monitor registers from EL0, EL1 and EL2 to EL3, from both Security states and both Execution states are trapped as follows:

- In AArch64 state, accesses to the following registers are trapped to EL3 and are reported using EC syndrome value 0x18:
  - PMCR_EL0, PMCNTENSET_EL0, PMCNTENCLR_EL0, PMOVSCLR_EL0, PMSWINC_EL0, PMSELR_EL0, PMCEID0_EL0, PMCEID1_EL0, PMCCNTR_EL0, PMXEVTYPER_EL0, PMXEVCNTR_EL0, PMUSERENR_EL0, PMINTENSET_EL1, PMINTENCLR_EL1, PMOVSSET_EL0, PMEVNCNTR<\text{n}>_EL0, PMEVTPYPER<\text{n}>_EL0.

- In AArch32 state, accesses using MCR or MRC to the following registers are reported using EC syndrome value 0x03, accesses using MCRR or MRRC are reported using EC syndrome value 0x04:
  - PMCR, PMCNTENSET, PMCNTENCLR, PMOVS, PMSWINC, PMSELR, PMCEID0, PMCEID1, PMCCNTR, PMXEVTYPER, PMXEVCNTR, PMUSERENR, PMINTENSET, PMINTENCLR, PMOVSSET, PMEVNCNTR<\text{n}>, PMEVTPYPER<\text{n}>, PMCFILTR.

- If ARMv8.4-PMU is implemented, in AArch64 state, PMMIR_EL1, and in AArch32 state, PMMIR.
- If ARMv8.1-PMU is implemented, in AArch32 state, PMCEID2, and PMCEID3.

<table>
<thead>
<tr>
<th>TPM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2, EL1, and EL0 System register accesses to all Performance Monitor registers are trapped to EL3, unless it is trapped by HDCR.TPM or MDCR_EL2.TPM.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

Accessing the MDCR_EL3

Accesses to this register use the following encodings:

MRS <Xt>, MDCR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return MDCR_EL3;

MSR MDCR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  MDCR_EL3 = X[t];
The MDRAR_EL1 characteristics are:

**Purpose**

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Armv8 deprecates any use of this register.

**Configuration**

AArch64 System register MDRAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register DBGDRAR[63:0].

**Attributes**

MDRAR_EL1 is a 64-bit register.

**Field descriptions**

The MDRAR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51:48</td>
<td>ROMADDR[51:48], bits [51:48]</td>
</tr>
<tr>
<td>47:12</td>
<td>ROMADDR[47:12], bits [47:12]</td>
</tr>
<tr>
<td></td>
<td>Bits [47:12] of the ROM table physical address.</td>
</tr>
</tbody>
</table>

**Bits [63:52]**

Reserved, RES0.

**ROMADDR[51:48], bits [51:48]**

When Armv8.2-LPA is implemented:


Otherwise:

Reserved, RES0.

**ROMADDR[47:12], bits [47:12]**

Bits[47:12] of the ROM table physical address.

When Armv8.2-LPA is implemented, ROMADDR[51:48] forms the upper part of the address value. Otherwise, ROMADDR[51:48] is RES0.

If the physical address size in bits (PAsize) is less than 52, then the register bits corresponding to ROMADDR [51:PAsize] are RES0.

Bits [11:0] of the ROM table physical address are zero.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system that supports AArch32 at the highest implemented Exception level.
In an implementation that includes EL3, ROMADDR is an address in Non-secure memory. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure memory.

Bits [11:2]

Reserved, RES0.

Valid, bits [1:0]

This field indicates whether the ROM Table address is valid. The permitted values of this field are:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>ROM Table address is not valid. Software must ignore ROMADDR.</td>
</tr>
<tr>
<td>0b11</td>
<td>ROM Table address is valid.</td>
</tr>
</tbody>
</table>

Other values are reserved.

Accessing the MDRAR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, MDRAR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MDRAR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MDRAR_EL1;
elsif PSTATE.EL == EL3 then
  return MDRAR_EL1;
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The MDSCR_EL1 characteristics are:

**Purpose**

Main control register for the debug implementation.

**Configuration**

AArch64 System register MDSCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDSRext[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MDSCR_EL1 is a 64-bit register.

**Field descriptions**

The MDSCR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>TFO, bit [31]</td>
</tr>
<tr>
<td>62</td>
<td>RXfull, bit [30]</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TFO, bit [31]**

When ARMv8.4-Trace is implemented:

Trace Filter override. Used for save/restore of EDSCR.TFO.

When OLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TFO. Reads and writes of this bit are indirect accesses to EDSCR.TFO.

Accessing this field has the following behavior:

- When OLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OLSR_EL1.OSLK == 0, access to this field is **RO**.

**Otherwise:**

Reserved, RES0.

**RXfull, bit [30]**

Used for save/restore of EDSCR.RXfull.
When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, software must treat this bit as UNK/SBZP.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, this bit holds the value of $\text{EDSCR}.\text{RXfull}$. Reads and writes of this bit are indirect accesses to $\text{EDSCR}.\text{RXfull}$.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, access to this field is RW.
- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, access to this field is RO.

$\text{TXfull}$, bit [29]

Used for save/restore of $\text{EDSCR}.\text{TXfull}$.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, software must treat this bit as UNK/SBZP.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, this bit holds the value of $\text{EDSCR}.\text{TXfull}$. Reads and writes of this bit are indirect accesses to $\text{EDSCR}.\text{TXfull}$.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, access to this field is RW.
- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, access to this field is RO.

$\text{Bit [28]}$

Reserved, RES0.

$\text{RXO}$, bit [27]

Used for save/restore of $\text{EDSCR}.\text{RXO}$.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, software must treat this bit as UNK/SBZP.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, this bit holds the value of $\text{EDSCR}.\text{RXO}$. Reads and writes of this bit are indirect accesses to $\text{EDSCR}.\text{RXO}$.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, access to this field is RW.
- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, access to this field is RO.

$\text{TXU}$, bit [26]

Used for save/restore of $\text{EDSCR}.\text{TXU}$.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, software must treat this bit as UNK/SBZP.

When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, this bit holds the value of $\text{EDSCR}.\text{TXU}$. Reads and writes of this bit are indirect accesses to $\text{EDSCR}.\text{TXU}$.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 1$, access to this field is RW.
- When $\text{OSLSR}_{\text{EL1}}.\text{OSLK} == 0$, access to this field is RO.
**Bits [25:24]**

Reserved, RES0.

**INTdis, bits [23:22]**

Used for save/restore of EDSCR.INTdis.

When OSLSR_EL1.OSLK == 0, and software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this field holds the value of EDSCR.INTdis. Reads and writes of this field are indirect accesses to EDSCR.INTdis.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

**TDA, bit [21]**

Used for save/restore of EDSCR.TDA.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.TDA. Reads and writes of this bit are indirect accesses to EDSCR.TDA.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

**Bit [20]**

Reserved, RES0.

**SC2, bit [19]**

When ARMv8.0-PCS is implemented, ARMv8.1-VHE is implemented and ARMv8.2-PCS is not implemented:

Used for save/restore of EDSCR.SC2.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.SC2. Reads and writes of this bit are indirect accesses to EDSCR.SC2.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

**Bits [18:16]**

Reserved, RAZ/WI.
Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a
read-modify-write sequence to write to the register.

**MDE, bit [15]**

Monitor debug events. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

<table>
<thead>
<tr>
<th>MDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Breakpoint, Watchpoint, and Vector Catch exceptions disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Breakpoint, Watchpoint, and Vector Catch exceptions enabled.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**HDE, bit [14]**

Used for save/restore of EDSCR.HDE.

When OSLSR_EL1.OSLK == 0, software must treat this bit as UNK/SBZP.

When OSLSR_EL1.OSLK == 1, this bit holds the value of EDSCR.HDE. Reads and writes of this bit are indirect
accesses to EDSCR.HDE.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is RW.
- When OSLSR_EL1.OSLK == 0, access to this field is RO.

**KDE, bit [13]**

Local (kernel) debug enable. If EL_D is using AArch64, enable debug exceptions within EL_D. Permitted values are:

<table>
<thead>
<tr>
<th>KDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Debug exceptions, other than Breakpoint Instruction exceptions,</td>
</tr>
<tr>
<td></td>
<td>disabled within EL_D.</td>
</tr>
<tr>
<td>0b1</td>
<td>All debug exceptions enabled within EL_D.</td>
</tr>
</tbody>
</table>

RES0 if EL_D is using AArch32.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**TDCC, bit [12]**

Traps EL0 accesses to the Debug Communication Channel (DCC) registers to EL1, or to EL2 when it is implemented
and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, as follows:

- In AArch64 state, MRS or MSR accesses to the following DCC registers are trapped, reported using EC
  syndrome value 0x18:
  - MDCCSR_EL0.
  - If not in Debug state, DBGDTR_EL0, DBGDTRTX_EL0, and DBGDTRRX_EL0.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped, reported using EC syndrome
  value 0x05:
  - DBGDSCRint, DBGDIDR, DBGDSAR, DBGDRAR.
  - If not in Debug state, DBGDTRRXint, and DBGDTRTXint.
- In AArch32 state, LDC access to DBGDTRRXint and STC access to DBGDTRTXint are trapped, reported using
  EC syndrome value 0x06.
- In AArch32 state, MRRC accesses to DBGDSAR and DBGDRAR are trapped, reported using EC syndrome
  value 0x0C.

<table>
<thead>
<tr>
<th>TDCC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>
| 0b1   | EL0 using AArch64: EL0 accesses to the AArch64 DCC registers are trapped.
EL0 using AArch32: EL0 accesses to the AArch32 DCC registers are trapped. |
On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [11:7]**

Reserved, RES0.

**ERR, bit [6]**

Used for save/restore of **EDSCR**.ERR.

When **OSLSR_EL1**.OSLK == 0, software must treat this bit as UNK/SBZP.

When **OSLSR_EL1**.OSLK == 1, this bit holds the value of **EDSCR**.ERR. Reads and writes of this bit are indirect accesses to **EDSCR**.ERR.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When **OSLSR_EL1**.OSLK == 1, access to this field is **RW**.
- When **OSLSR_EL1**.OSLK == 0, access to this field is **RO**.

**Bits [5:1]**

Reserved, RES0.

**SS, bit [0]**

Software step control bit. If ELD is using AArch64, enable Software step. Permitted values are:

<table>
<thead>
<tr>
<th>SS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Software step disabled</td>
</tr>
<tr>
<td>0b1</td>
<td>Software step enabled</td>
</tr>
</tbody>
</table>

RES0 if ELD is using AArch32.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the MDSCR_EL1**

Individual fields within this register might have restricted accessibility when **OSLSR_EL1**.OSLK == 0 (the OS lock is unlocked). See the field descriptions for more detail.

Accesses to this register use the following encodings:

**MRS <Xt>, MDSCR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.MDSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x158];
    else
        return MDSCR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MDSCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MDSCR_EL1 = X[t];
    else
        MDSCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MDSCR_EL1 = X[t];
else
    MDSCR_EL1 = X[t];
end if

MMR MDSCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.MDSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x158] = X[t];
    else
        MDSCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MDSCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MDSCR_EL1 = X[t];
else
    MDSCR_EL1 = X[t];
end if
**MIDR_EL1, Main ID Register**

The MIDR_EL1 characteristics are:

**Purpose**

Provides identification information for the PE, including an implementer code for the device and a device ID number.

**Configuration**

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR[31:0].

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to External register MIDR_EL1[31:0].

**Attributes**

MIDR_EL1 is a 64-bit register.

**Field descriptions**

The MIDR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Bits [63:32]

Reserved, RES0.

**Implementer, bits [31:24]**

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

<table>
<thead>
<tr>
<th>Hex representation</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved for software use</td>
</tr>
<tr>
<td>0xC0</td>
<td>Ampere Computing</td>
</tr>
<tr>
<td>0x41</td>
<td>Arm Limited</td>
</tr>
<tr>
<td>0x42</td>
<td>Broadcom Corporation</td>
</tr>
<tr>
<td>0x43</td>
<td>Cavium Inc.</td>
</tr>
<tr>
<td>0x44</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>0x46</td>
<td>Fujitsu Ltd.</td>
</tr>
<tr>
<td>0x49</td>
<td>Infineon Technologies AG</td>
</tr>
<tr>
<td>0x4D</td>
<td>Motorola or Freescale Semiconductor Inc.</td>
</tr>
<tr>
<td>0x4E</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>0x50</td>
<td>Applied Micro Circuits Corporation</td>
</tr>
<tr>
<td>0x51</td>
<td>Qualcomm Inc.</td>
</tr>
<tr>
<td>0x56</td>
<td>Marvell International Ltd.</td>
</tr>
<tr>
<td>0x69</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.
**Variant, bits [23:20]**

An implementation defined variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

**Architecture, bits [19:16]**

The permitted values of this field are:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Armv4.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Armv4T.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Armv5 (obsolete).</td>
</tr>
<tr>
<td>0b0100</td>
<td>Armv5T.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Armv5TE.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Armv5TEJ.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv6.</td>
</tr>
</tbody>
</table>
| 0b1111       | Architectural features are individually identified in the ID_* registers, see 'ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K12.3.3. All other values are reserved.

**PartNum, bits [15:4]**

An implementation defined primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

**Revision, bits [3:0]**

An implementation defined revision number for the device.

**Accessing the MIDR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, MIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b000</td>
<td>0b000</td>
<td>0b00</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGTR_EL2.MIDR_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
      return VPIDR_EL2;
    else
      return MIDR_EL1;
  elsif PSTATE.EL == EL2 then
    return MIDR_EL1;
  elsif PSTATE.EL == EL3 then
    return MIDR_EL1;
The MPAM0_EL1 characteristics are:

**Purpose**

Holds information to generate MPAM labels for memory requests when executing at EL0. When EL2 is present and enabled, the MPAM virtualization option is present, MPAMHCR_EL2.GSTAPP.PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM information to label memory requests.

If EL2 is present and enabled, and HCR_EL2.E2H == 0 or HCR_EL2.TGE == 0, the MPAM virtualization option is present and MPAMHCR_EL2.EL0.VPMEN == 1, then MPAM PARTIDs in MPAM0_EL1 are virtual and mapped into physical PARTIDs for the current Security state.

**Configuration**

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM0_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MPAM0_EL1 is a 64-bit register.

**Field descriptions**

The MPAM0_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | PMG_D | PMG_I | PARTID_D | PARTID_I |

**Bits [63:48]**

Reserved, RES0.

**PMG_D, bits [47:40]**

Performance monitoring group property for PARTID_D.

This field resets to an architecturally UNKNOWN value.

**PMG_I, bits [39:32]**

Performance monitoring group property for PARTID_I.

This field resets to an architecturally UNKNOWN value.

**PARTID_D, bits [31:16]**

Partition ID for data accesses, including load and store accesses, made from EL0.

This field resets to an architecturally UNKNOWN value.
PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL0.

This field resets to an architecturally UNKNOWN value.

Accessing the MPAM0_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

MRS <Xt>, MPAM0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsiif PSTATE.EL == EL1 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x10);
  else
    return MPAM0_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  else
    MPAM0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  MPAM0_EL1 = X[t];

MSR MPAM0_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsiif PSTATE.EL == EL1 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x10);
  else
    MPAM0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  else
    MPAM0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  MPAM0_EL1 = X[t];
The MPAM1_EL1 characteristics are:

**Purpose**

Holds information to generate MPAM labels for memory requests when executing at EL1.

When EL2 is present and enabled, the MPAM virtualization option is present. MPAMHCR_EL2, GSTAPP PLK == 1 and HCR_EL2, TGE == 0, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests when executing at EL0.

MPAM1_EL1 is an alias for MPAM2_EL2 when executing at EL2 with HCR_EL2, E2H == 1.

MPAM1_EL12 is an alias for MPAM1_EL1 when executing at EL2 or EL3 with HCR_EL2, E2H == 1.

If EL2 is present and enabled, the MPAM virtualization option is present and MPAMHCR_EL2, EL1_VPMEN == 1, MPAM PARTIDs in MPAM1_EL1 are virtual and mapped into physical PARTIDs for the current Security state. This mapping of MPAM1_EL1 virtual PARTIDs to physical PARTIDs when EL1_VPMEN is 1 also applies when MPAM1_EL1 is used at EL0 due to MPAMHCR_EL2, GSTAPP PLK.

**Configuration**

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when HaveEL(EL3).

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register MPAM2_EL2[63] when !HaveEL(EL3) and HaveEL(EL2).

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM1_EL1 are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAM1_EL1 is a 64-bit register.

**Field descriptions**

The MPAM1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>MPAMEN, bit [63] MPAM Enable</td>
<td>0b0</td>
<td>The default PARTID and default PMG are output in MPAM information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0b1</td>
<td>MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.</td>
</tr>
</tbody>
</table>

If neither EL3 nor EL2 is implemented, this field is read/write.
If EL3 is implemented, this field is read-only and reads the current value of the read/write bit MPAM3_EL3.MPAMEN.

If EL3 is not implemented and EL2 is implemented, this field is read-only and reads the current value of the read/write bit MPAM2_EL2.MPAMEN.

This field resets to 0.

Accessing this field has the following behavior:

- When !HaveEL(EL3) and !HaveEL(EL2), access to this field is RW.
- Otherwise, access to this field is RO.

Bits [62:61]

Reserved, RES0.

FORCED_NS, bit [60]

When ARMv8.6-MPAM is implemented:

In the Secure state, FORCED_NS Secure state indicates the state of MPAM3_EL3.FORCE_NS.

<table>
<thead>
<tr>
<th>FORCED_NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>In the Non-secure state, always reads as 0. In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Indicate that MPAM3_EL3.FORCE_NS == 1.</td>
</tr>
</tbody>
</table>

Reads as 0 in the Non-secure state.

Writes are ignored.

Otherwise:

Reserved, RES0.

Bits [59:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

This field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

This field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL1.

This field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

This field resets to an architecturally UNKNOWN value.
Accessing the MPAM1_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, accesses from EL3 using the mnemonic MPAM1_EL1 or MPAM1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

MRS <Xt>, MPAM1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x900];
    else
        return MPAM1_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return MPAM2_EL2;
        else
            return MPAM1_EL1;
    elsif PSTATE.EL == EL3 then
        return MPAM1_EL1;

MSR MPAM1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x900] = X[t];
    else
        MPAM1_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            MPAM2_EL2 = X[t];
        else
            MPAM1_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAM1_EL1 = X[t];
MRS <Xt>, MPAM1_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x900];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        MPAM1_EL1 = <Xt>;
      else
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
        MPAM1_EL1 = <Xt>;
      else
        UNDEFINED;
    else
      UNDEFINED;
end if

MSR MPAM1_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x900] = <Xt>;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
      if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        MPAM1_EL1 = <Xt>;
      else
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
        MPAM1_EL1 = <Xt>;
      else
        UNDEFINED;
  else
    UNDEFINED;
else
  UNDEFINED;
end if
The MPAM2_EL2 characteristics are:

**Purpose**

Holds information to generate MPAM labels for memory requests when executing at EL2.

**Configuration**

AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register MPAM3_EL3[63] when HaveEL(EL3).

AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register MPAM1_EL1[63].

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MPAM2_EL2 is a 64-bit register.

**Field descriptions**

The MPAM2_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>MPAMEN</th>
<th>RES0</th>
<th>TIDR</th>
<th>RES0</th>
<th>TRAPMPAM0EL1</th>
<th>TRAPMPAM1EL1</th>
<th>PMG_D</th>
<th>PMG_I</th>
<th>PARTID_D</th>
<th>PARTID_I</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
</tbody>
</table>

**MPAMEN, bit [63]**

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

<table>
<thead>
<tr>
<th>MPAMEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The default PARTID and default PMG are output in MPAM information from all Exception levels.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write MPAM3_EL3.MPAMEN bit.

This field resets to 0.

Accessing this field has the following behavior:

- When !HaveEL(EL3), access to this field is RW.
- Otherwise, access to this field is RO.
Bits [62:59]

Reserved, RES0.

**TIDR, bit [58]**

When ARMv8.6-MPAM is implemented and MPAMIDR_EL1.HAS_TIDR == 1:

TIDR traps accesses to MPAMIDR_EL1 from EL1 to EL2.

<table>
<thead>
<tr>
<th>TIDR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trap accesses to <strong>MPAMIDR_EL1</strong> from EL1 to EL2.</td>
</tr>
</tbody>
</table>

MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == 1 also traps **MPAMIDR_EL1** accesses from EL1 to EL2. If either TIDR or TRAP_MPAMIDR_EL1 are 1, accesses are trapped.

Otherwise:

Reserved, RES0.

Bits [57:50]

Reserved, RES0.

**TRAPMPAM0EL1, bit [49]**

TRAPMPAM0EL1: Trap accesses from EL1 to the **MPAM0_EL1** register trap to EL2.

<table>
<thead>
<tr>
<th>TRAPMPAM0EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to <strong>MPAM0_EL1</strong> from EL1 are not trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to <strong>MPAM0_EL1</strong> from EL1 are trapped to EL2.</td>
</tr>
</tbody>
</table>

When EL3 is not implemented, this field resets to 1. When EL3 is implemented, this field resets to **UNKNOWN**.

**TRAPMPAM1EL1, bit [48]**

TRAPMPAM1EL1: Trap accesses from EL1 to the **MPAM1_EL1** register trap to EL2.

<table>
<thead>
<tr>
<th>TRAPMPAM1EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to <strong>MPAM1_EL1</strong> from EL1 are not trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to <strong>MPAM1_EL1</strong> from EL1 are trapped to EL2.</td>
</tr>
</tbody>
</table>

When EL3 is not implemented, this field resets to 1. When EL3 is implemented, this field resets to **UNKNOWN**.

**PMG_D, bits [47:40]**

Performance monitoring group for data accesses.

This field resets to an architecturally **UNKNOWN** value.

**PMG_I, bits [39:32]**

Performance monitoring group for instruction accesses.

This field resets to an architecturally **UNKNOWN** value.
PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL2.
This field resets to an architecturally *UNKNOWN* value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.
This field resets to an architecturally *UNKNOWN* value.

**Accessing the MPAM2_EL2**

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ MPAM2_EL2}
\]

\[
\begin{array}{c|c|c|c|c}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b100 & 0b1010 & 0b0101 & 0b000 \\
\end{array}
\]

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAM2_EL2;
elsif PSTATE.EL == EL3 then
  return MPAM2_EL2;

\[
\text{MSR MPAM2_EL2, } <Xt>
\]

\[
\begin{array}{c|c|c|c|c}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b100 & 0b1010 & 0b0101 & 0b000 \\
\end{array}
\]
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    MPAM2_EL2 = X[t];
  end
elsif PSTATE.EL == EL3 then
  MPAM2_EL2 = X[t];
end

MRS <Xt>, MPAM1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x900];
  else
    return MPAM1_EL1;
  end
 elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    return MPAM2_EL2;
  else
    return MPAM1_EL1;
  end
elsif PSTATE.EL == EL3 then
  return MPAM1_EL1;
end

MSR MPAM1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x900] = X[t];
    else
        MPAM1_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            MPAM2_EL2 = X[t];
        else
            MPAM1_EL1 = X[t];
        elsif PSTATE.EL == EL3 then
            MPAM1_EL1 = X[t];
else
    MPAM1_EL1 = X[t];
The MPAM_EL3 characteristics are:

**Purpose**

Holds information to generate MPAM labels for memory requests when executing at EL3.

**Configuration**

AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register `MPAM2_EL2[63]` when HaveEL(EL2).

AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register `MPAM1_EL1[63]`.

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAM3_EL3 are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAM3_EL3 is a 64-bit register.

**Field descriptions**

The MPAM_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| MPAMEN | TRAPLOWER | SDEFLT | FORCE_NS | RES0 | PMG_D | PMG_I | PARTID_D | PARTID_I |
| 0b0 | The default PARTID and default PMG are output in MPAM information when executing at any ELn. |
| 0b1 | MPAM information is output based on the MPAn_ELx register for ELn according the MPAM configuration. |

This field is always read/write in MPAM3_EL3.

This field resets to 0.

**MPAMEN, bit [63]**

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

<table>
<thead>
<tr>
<th>MPAMEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The default PARTID and default PMG are output in MPAM information when executing at any ELn.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration.</td>
</tr>
</tbody>
</table>

This field is always read/write in MPAM3_EL3.

**TRAPLOWER, bit [62]**

Trap direct accesses to any MPAM system registers that are not **UNDEFINED** from all ELn lower than EL3.
**TRAPLOWER**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not force trapping of direct accesses of MPAM system registers to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>Force all direct accesses of MPAM system registers to trap to EL3.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 1.

**SDEFLT, bit [61]**

When ARMv8.6-MPAM is implemented and MPAMIDR_EL1.HAS_SDEFLT == 1:

SDEFLT overrides the PARTID with the default PARTID when executing in the Secure state.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PARTID is determined normally in the Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PARTID is always PARTID 0 when executing in the Secure state.</td>
</tr>
</tbody>
</table>

This field resets to an UNKNOWN value.

Otherwise:

Reserved, RES0.

**FORCE_NS, bit [60]**

When ARMv8.6-MPAM is implemented and MPAMIDR_EL1.HAS_FORCE_NS == 1:

FORCE_NS forces MPAM_NS to always be 1 in the Secure state.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAM_NS is 0 when executing in the Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM_NS is 1 when executing in the Secure state.</td>
</tr>
</tbody>
</table>

This field resets to an UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [59:48]**

Reserved, RES0.

**PMG_D, bits [47:40]**

Performance monitoring group for data accesses.

This field resets to an architecturally UNKNOWN value.

**PMG_I, bits [39:32]**

Performance monitoring group for instruction accesses.

**PARTID_D, bits [31:16]**

Partition ID for data accesses, including load and store accesses, made from EL3.

This field resets to an architecturally UNKNOWN value.
PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL3.
This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAM3_EL3**

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

MRS <Xt>, MPAM3_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MPAM3_EL3;

MSR MPAM3_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MPAM3_EL3 = X[t];
The MPAMHCR_EL2 characteristics are:

**Purpose**

Controls the PARTID virtualization features of MPAM. It controls the mapping of virtual PARTIDs into physical PARTIDs in MPAM0_EL1 when EL0_VPMEN == 1 and in MPAM1_EL1 when EL1_VPMEN == 1.

**Configuration**

This register is present only when MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMHCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MPAMHCR_EL2 is a 64-bit register.

**Field descriptions**

The MPAMHCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62616059585756555453525150494847464544434241</td>
</tr>
<tr>
<td>62</td>
<td>40</td>
</tr>
<tr>
<td>61</td>
<td>393837363534</td>
</tr>
<tr>
<td>60</td>
<td>33</td>
</tr>
<tr>
<td>59</td>
<td>32</td>
</tr>
<tr>
<td>58</td>
<td>RES0</td>
</tr>
<tr>
<td>57</td>
<td>TRAP_MPAMIDR_EL1</td>
</tr>
<tr>
<td>56</td>
<td>RES0</td>
</tr>
<tr>
<td>55</td>
<td>GSTAPP_PLK</td>
</tr>
<tr>
<td>54</td>
<td>RES0</td>
</tr>
<tr>
<td>53</td>
<td>EL1_VPMEN</td>
</tr>
<tr>
<td>52</td>
<td>EL0_VPMEN</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TRAP_MPAMIDR_EL1, bit [31]**

Trap accesses from EL1 to MPAMIDR_EL1 to EL2.

<table>
<thead>
<tr>
<th>TRAP_MPAMIDR_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Direct accesses to MPAMIDR_EL1 from EL1 are trapped to EL2.</td>
</tr>
</tbody>
</table>

When EL3 is not implemented, this field is reset to 1. When EL3 is implemented, this field resets to UNKNOWN.

**Bits [30:9]**

Reserved, RES0.
GSTAPP_PLK, bit [8]

Make the PARTIDs at EL0 the same as the PARTIDs at EL1. When executing at EL0, EL2 is enabled, HCR_EL2.TGE == 0 and GSTAPP_PLK = 1, MPAM1_EL1 is used instead of MPAM0_EL1 to generate MPAM labels for memory requests.

<table>
<thead>
<tr>
<th>GSTAPP_PLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAM0_EL1 is used to generate MPAM labels when executing at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM1_EL1 is used to generate MPAM labels when executing at EL0 with EL2 enabled and HCR_EL2.TGE == 0. Otherwise MPAM0_EL1 is used.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bits [7:2]

Reserved, RES0.

EL1_VPMEN, bit [1]

Enable the virtual PARTID mapping of the PARTID fields in MPAM1_EL1 when executing at EL1. This bit also enables virtual PARTID mapping when MPAM1_EL1 is used to generate MPAM labels for memory requests at EL0 due to GSTAPP_PLK == 1.

<table>
<thead>
<tr>
<th>EL1_VPMEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are physical PARTIDs that are used to label memory system requests.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM1_EL1.PARTID_I and MPAM1_EL1.PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

EL0_VPMEN, bit [0]

Enable the virtual PARTID mapping of the PARTID fields of MPAM0_EL1 unless HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1.

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1, EL0_VPMEN is ignored and MPAM0_EL1 PARTID fields are not mapped.

When MPAMCHR_EL2, GSTAPP_PLK == 1 and HCR_EL2.TGE == 0, MPAM1_EL1 is used as the source of PARTIDs and the virtual PARTID mapping of MPAM1_EL1 PARTIDs is controlled by MPAMCHR_EL2.EL1_VPMEN.

<table>
<thead>
<tr>
<th>EL0_VPMEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are physical PARTIDs that are used to label memory system requests.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM0_EL1.PARTID_I and MPAM0_EL1.PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Accessing the MPAMCHR_EL2

Accesses to this register use the following encodings:
MRS <Xt>, MPAMHCR_EL2

### MPAMHCR_EL2, MPAM Hypervisor Control Register (EL2)

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x930];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return MPAMHCR_EL2;
  elsif PSTATE.EL == EL3 then
    return MPAMHCR_EL2;
  fi

MSR MPAMHCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x930] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      MPAMHCR_EL2 = X[t];
  elsif PSTATE.EL == EL3 then
    MPAMHCR_EL2 = X[t];
  fi

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MPAMIDR_EL1, MPAM ID Register (EL1)

The MPAMIDR_EL1 characteristics are:

**Purpose**

Indicates the presence and maximum PARTID and PMG values supported in the implementation. It also indicates whether the implementation supports MPAM virtualization.

**Configuration**

This register is present only when MPAM is implemented. Otherwise, direct accesses to MPAMIDR_EL1 are **UNDEFINED**.

**Attributes**

MPAMIDR_EL1 is a 64-bit register.

**Field descriptions**

The MPAMIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>HAS_SDEFLT, bit [61]</td>
</tr>
<tr>
<td>60:59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>VPMR_MAX</td>
</tr>
<tr>
<td>57</td>
<td>HAS_HCR</td>
</tr>
<tr>
<td>56:52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51:48</td>
<td>PARTID_MAX</td>
</tr>
<tr>
<td>47:4</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

MPAMIDR_EL1 indicates the MPAM implementation parameters of the PE.

**Bits [63:62]**

Reserved, RES0.

**HAS_SDEFLT, bit [61]**

From Armv8.6:

HAS_SDEFLT indicates support for MPAM3_EL3.SDEFLT bit.

<table>
<thead>
<tr>
<th>HAS_SDEFLT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The SDEFLT bit is not implemented in MPAM3_EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>The SDEFLT bit is implemented in MPAM3_EL3.</td>
</tr>
</tbody>
</table>

When MPAM3_EL3.SDEFLT == 1, accesses from the Secure execution state use the default PARTID, PARTID == 0.

**Otherwise:**

Reserved, RES0.

**HAS_FORCE_NS, bit [60]**

From Armv8.6:

HAS_FORCE_NS indicates support for MPAM3_EL3.Force_NS bit.
### HAS_FORCE_NS

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0  The FORCE_NS bit is not implemented in <strong>MPAM3_EL3</strong>.</td>
</tr>
<tr>
<td>0b1  The FORCE_NS bit is implemented in <strong>MPAM3_EL3</strong>.</td>
</tr>
</tbody>
</table>

When **MPAM3_EL3**.FORCE_NS == 1, accesses from the Secure execution state have MPAM_NS == 1.

Otherwise:

Reserved, RES0.

### Bit [59]

Reserved, RES0.

### HAS_TIDR, bit [58]

*From Armv8.6:*

HAS_TIDR indicates support for **MPAM2_EL2**.TIDR bit.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0  The TIDR bit is not implemented in <strong>MPAM2_EL2</strong>.</td>
</tr>
<tr>
<td>0b1  The TIDR bit is implemented in <strong>MPAM2_EL2</strong>.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

### Bits [57:40]

Reserved, RES0.

### PMG_MAX, bits [39:32]

The largest value of PMG that the implementation can generate. The PMG_I and PMG_D fields of every MPAMn_ELx must implement at least enough bits to represent PMG_MAX.

### Bits [31:21]

Reserved, RES0.

### VPMR_MAX, bits [20:18]

If HAS_HCR == 0, VPMR_MAX must be 0b000. Otherwise, it indicates the maximum register index n for the MPAMVPM<n>_EL2 registers.

### HAS_HCR, bit [17]

HAS_HCR indicates that the PE implementation supports MPAM virtualization, including **MPAMHCR_EL2**, **MPAMVPMV_EL2** and **MPAMVPM<n>_EL2** with n in the range 0 to VPMR_MAX. Must be 0 if EL2 is not implemented in either security state.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0  MPAM virtualization is not supported.</td>
</tr>
<tr>
<td>0b1  MPAM virtualization is supported.</td>
</tr>
</tbody>
</table>
**Bit [16]**

Reserved, RES0.

**PARTID_MAX, bits [15:0]**

The largest value of PARTID that the implementation can generate. The PARTID_I and PARTID_D fields of every MPAMn_ELx must implement at least enough bits to represent PARTID_MAX.

**Accessing the MPAMIDR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, MPAMIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```javascript
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAMIDR_EL1.HAS_HCR == '1' &&
        MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MPAMIDR_EL1.HAS_TIDR == '1' &&
        MPAM2_EL2.TIDR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAMIDR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMIDR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return MPAMIDR_EL1;
```

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MPAMVPM0_EL2, MPAM Virtual PARTID Mapping Register 0

The MPAMVPM0_EL2 characteristics are:

**Purpose**

MPAMVPM0_EL2 provides mappings from virtual PARTIDs 0 - 3 to physical PARTIDs.

The **MPAMIDR_EL1.VPMR_MAX** field gives the index of the highest implemented MPAMVPM<n>_EL2 register. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, **MPAMVPM0_EL2**.

Virtual PARTID mapping is enabled by **MPAMHCR_EL2.EL1_VPMEN** for PARTIDs in **MPAM1_EL1** and by **MPAMHCR_EL2.EL0_VPMEN** for PARTIDs in **MPAM0_EL1**.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the **MPAMVPMV_EL2.VPM_V** bit in bit position n is set to 1.

**Configuration**

This register is present only when MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPM0_EL2 are **UNDEFINED**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAMVPM0_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPM0_EL2 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40 | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| PhyPARTID3 |
| PhyPARTID2 |
| PhyPARTID1 |
| PhyPARTID0 |

**PhyPARTID3, bits [63:48]**

Virtual PARTID Mapping Entry for virtual PARTID 3. PhyPARTID3 gives the mapping of virtual PARTID 3 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID2, bits [47:32]**

Virtual PARTID Mapping Entry for virtual PARTID 2. PhyPARTID2 gives the mapping of virtual PARTID 2 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.
**PhyPARTID1, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 1. PhyPARTID1 gives the mapping of virtual PARTID 1 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID0, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 0. PhyPARTID0 gives the mapping of virtual PARTID 0 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM0_EL2**

Accesses to this register use the following encodings:

\[
\text{MRS } <Xt>, \text{ MPAMVPM0_EL2}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x940];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAMVPM0_EL2;
  end
elsif PSTATE.EL == EL3 then
  return MPAMVPM0_EL2;

\[
\text{MSR } \text{MPAMVPM0_EL2, } <Xt>
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x940] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    MPAMVPM0_EL2 = X[t];
  end
elsif PSTATE.EL == EL3 then
  MPAMVPM0_EL2 = X[t];
The MPAMVPM1_EL2 characteristics are:

### Purpose

MPAMVPM1_EL2 provides mappings from virtual PARTIDs 4 - 7 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

### Configuration

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 0. Otherwise, direct accesses to MPAMVPM1_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

MPAMVPM1_EL2 is a 64-bit register.

### Field descriptions

The MPAMVPM1_EL2 bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>PhyPARTID7</th>
<th>PhyPARTID6</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59-56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55-52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51-48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47-44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43-40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39-36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35-32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31-28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27-24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23-20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19-16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15-12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11-8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7-4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3-0</td>
<td>PhyPARTID5</td>
<td>PhyPARTID4</td>
</tr>
</tbody>
</table>

**PhyPARTID7, bits [63:48]**

Virtual PARTID Mapping Entry for virtual PARTID 7. PhyPARTID7 gives the mapping of virtual PARTID 7 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

**PhyPARTID6, bits [47:32]**

Virtual PARTID Mapping Entry for virtual PARTID 6. PhyPARTID6 gives the mapping of virtual PARTID 6 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.
**PhyPARTID5, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 5. PhyPARTID5 gives the mapping of virtual PARTID 5 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID4, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 4. PhyPARTID4 gives the mapping of virtual PARTID 4 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM1_EL2**

Accesses to this register use the following encodings:

```plaintext
MRS <Xt>, MPAMVPM1_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x948];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAMVPM1_EL2;
  end
elsif PSTATE.EL == EL3 then
  return MPAMVPM1_EL2;
```

```plaintext
MSR MPAMVPM1_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x948] = X[t];
    elif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM1_EL2 = X[t];
    end
elsif PSTATE.EL == EL3 then
    MPAMVPM1_EL2 = X[t];

The MPAMVPM2_EL2 characteristics are:

### Purpose

MPAMVPM2_EL2 provides mappings from virtual PARTIDs 8 - 11 to physical PARTIDs.

- **MPAMIDR_EL1.VPMR_MAX** field gives the index of the highest implemented MPAMVPM0_EL2 to MPAMVPM7_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

### Configuration

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 1. Otherwise, direct accesses to MPAMVPM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

MPAMVPM2_EL2 is a 64-bit register.

### Field descriptions

The MPAMVPM2_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>PhyPARTID11</td>
<td>PhyPARTID9</td>
<td>PhyPARTID10</td>
<td>PhyPARTID8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**PhyPARTID11, bits [63:48]**

Virtual PARTID Mapping Entry for virtual PARTID 11. PhyPARTID11 gives the mapping of virtual PARTID 11 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

**PhyPARTID10, bits [47:32]**

Virtual PARTID Mapping Entry for virtual PARTID 10. PhyPARTID10 gives the mapping of virtual PARTID 10 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.
**PhyPARTID9, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 9. PhyPARTID9 gives the mapping of virtual PARTID 9 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID8, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 8. PhyPARTID8 gives the mapping of virtual PARTID 8 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM2_EL2**

Accesses to this register use the following encodings:

```plaintext
MRS <Xt>, MPAMVPM2_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x950];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAMVPM2_EL2;
elsif PSTATE.EL == EL3 then
  return MPAMVPM2_EL2;
```

```plaintext
MSR MPAMVPM2_EL2, <Xt>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x950] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    MPAMVPM2_EL2 = X[t];
  end
elsif PSTATE.EL == EL3 then
  MPAMVPM2_EL2 = X[t];
The MPAMVPM3_EL2 characteristics are:

**Purpose**

MPAMVPM3_EL2 provides mappings from virtual PARTIDs 12 - 15 to physical PARTIDs.

MPAMIDR_EL1.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, MPAMVPM0_EL2.

Virtual PARTID mapping is enabled by MPAMHCR_EL2.EL1_VPMEN for PARTIDs in MPAM1_EL1 and by MPAMHCR_EL2.EL0_VPMEN for PARTIDs in MPAM0_EL1.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the MPAMVPMV_EL2.VPM_V bit in bit position n is set to 1.

**Configuration**

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 2. Otherwise, direct accesses to MPAMVPM3_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MPAMVPM3_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPM3_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>PhyPARTID15</th>
<th>PhyPARTID14</th>
<th>PhyPARTID13</th>
<th>PhyPARTID12</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48</td>
<td>47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16</td>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**PhyPARTID15, bits [63:48]**

Virtual PARTID Mapping Entry for virtual PARTID 15. PhyPARTID15 gives the mapping of virtual PARTID 15 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

**PhyPARTID14, bits [47:32]**

Virtual PARTID Mapping Entry for virtual PARTID 14. PhyPARTID14 gives the mapping of virtual PARTID 14 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.
**PhyPARTID13, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 13. PhyPARTID13 gives the mapping of virtual PARTID 13 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

**PhyPARTID12, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 12. PhyPARTID12 gives the mapping of virtual PARTID 12 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

**Accessing the MPAMVPM3_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, MPAMVPM3_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x958];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAMVPM3_EL2;
else
  PSTATE.EL == EL3 then
    return MPAMVPM3_EL2;

MSR MPAMVPM3_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x958] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    endelse
  endelse
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    MPAMVPM3_EL2 = X[t];
  endelse
elsif PSTATE.EL == EL3 then
  MPAMVPM3_EL2 = X[t];
The MPAMVPM4_EL2 characteristics are:

**Purpose**

MPAMVPM4_EL2 provides mappings from virtual PARTIDs 16 - 19 to physical PARTIDs.

**MPAMIDR_EL1**.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, **MPAMVPM0_EL2**.

Virtual PARTID mapping is enabled by **MPAMHCR_EL2.EL1_VPMEN** for PARTIDs in **MPAM1_EL1** and by **MPAMHCR_EL2.EL0_VPMEN** for PARTIDs in **MPAM0_EL1**.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the **MPAMVPMV_EL2**.VPM_V bit in bit position n is set to 1.

**Configuration**

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 3. Otherwise, direct accesses to MPAMVPM4_EL2 are **UNDEFINED**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAMVPM4_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPM4_EL2 bit assignments are:

|       | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| PhyPARTID19 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| PhyPARTID18 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**PhyPARTID19**, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 19. PhyPARTID19 gives the mapping of virtual PARTID 19 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID18**, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 18. PhyPARTID18 gives the mapping of virtual PARTID 18 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.
**PhyPARTID17, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 17. PhyPARTID17 gives the mapping of virtual PARTID 17 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID16, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 16. PhyPARTID16 gives the mapping of virtual PARTID 16 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM4_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, MPAMVPM4_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x960];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM4_EL2;
    end
elsif PSTATE.EL == EL3 then
    return MPAMVPM4_EL2;

MSR MPAMVPM4_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x960] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM4_EL2 = X[t];
    end
elsif PSTATE.EL == EL3 then
    MPAMVPM4_EL2 = X[t];

MPAMVPM5_EL2, MPAM Virtual PARTID Mapping Register 5

The MPAMVPM5_EL2 characteristics are:

**Purpose**

MPAMVPM5_EL2 provides mappings from virtual PARTIDs 20 - 23 to physical PARTIDs.

**MPAMIDR_EL1**.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If **MPAMIDR_EL1**.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, **MPAMVPM0_EL2**.

Virtual PARTID mapping is enabled by **MPAMHCR_EL2**.EL1_VPMEN for PARTIDs in **MPAM1_EL1** and by **MPAMHCR_EL2**.EL0_VPMEN for PARTIDs in **MPAM0_EL1**.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the **MPAMVPMV_EL2**.VPM_V bit in bit position n is set to 1.

**Configuration**

This register is present only when MPAM is implemented, **MPAMIDR_EL1**.HAS_HCR == 1 and **MPAMIDR_EL1**.VPMR_MAX > 4. Otherwise, direct accesses to MPAMVPM5_EL2 are **UNDEFINED**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAMVPM5_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPM5_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**PhyPARTID23**, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 23. PhyPARTID23 gives the mapping of virtual PARTID 23 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID22**, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 22. PhyPARTID22 gives the mapping of virtual PARTID 22 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.
**PhyPARTID21, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 21. PhyPARTID21 gives the mapping of virtual PARTID 21 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID20, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 20. PhyPARTID20 gives the mapping of virtual PARTID 20 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM5_EL2**

Accesses to this register use the following encodings:

```c
MRS <Xt>, MPAMVPM5_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b101</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  UNDEFINED;
elsesi PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2."NV2, NV" == '11' then
    return NVMem[0x968];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAMVPM5_EL2;
  endif
elsif PSTATE.EL == EL3 then
  return MPAMVPM5_EL2;
```

```c
MSR MPAMVPM5_EL2, <Xt>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x968] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    MPAMVPM5_EL2 = X[t];
  end
elsif PSTATE.EL == EL3 then
  MPAMVPM5_EL2 = X[t];
MPAMVPM6_EL2, MPAM Virtual PARTID Mapping Register 6

The MPAMVPM6_EL2 characteristics are:

**Purpose**

MPAMVPM6_EL2 provides mappings from virtual PARTIDs 24 - 27 to physical PARTIDs.

The **MPAMIDR_EL1**.VPMR_MAX field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If **MPAMIDR_EL1**.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, **MPAMVPM0_EL2**.

Virtual PARTID mapping is enabled by **MPAMHCR_EL2**.EL1_VPMEN for PARTIDs in **MPAM1_EL1** and by **MPAMHCR_EL2**.EL0_VPMEN for PARTIDs in **MPAM0_EL1**.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the **MPAMVPMV_EL2**.VPM_V bit in bit position n is set to 1.

**Configuration**

This register is present only when MPAM is implemented, **MPAMIDR_EL1**.HAS_HCR == 1 and **MPAMIDR_EL1**.VPMR_MAX > 5. Otherwise, direct accesses to MPAMVPM6_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MPAMVPM6_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPM6_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| PhyPARTID27 | PhyPARTID26 |
| PhyPARTID25 | PhyPARTID24 |

**PhyPARTID27**, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 27. PhyPARTID27 gives the mapping of virtual PARTID 27 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.

**PhyPARTID26**, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 26. PhyPARTID26 gives the mapping of virtual PARTID 26 to a physical PARTID.

This field resets to an architecturally UNKNOWN value.
**PhyPARTID25, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 25. PhyPARTID25 gives the mapping of virtual PARTID 25 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID24, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 24. PhyPARTID24 gives the mapping of virtual PARTID 24 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM6_EL2**

Accesses to this register use the following encodings:

```
MRS <Xt>, MPAMVPM6_EL2
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x970];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return MPAMVPM6_EL2;
  end
elsif PSTATE.EL == EL3 then
  return MPAMVPM6_EL2;
```

**MSR MPAMVPM6_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x970] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEl(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
    end
elseif PSTATE.EL == EL2 then
    if HaveEl(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM6_EL2 = X[t];
    end
elseif PSTATE.EL == EL3 then
    MPAMVPM6_EL2 = X[t];
The MPAMVPM7_EL2 characteristics are:

**Purpose**

MPAMVPM7_EL2 provides mappings from virtual PARTIDs 28 - 31 to physical PARTIDs.

- **MPAMIDR_EL1.VPMR_MAX** field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. 
- **VPMR_MAX** can be as large as 7 (8 registers) or 32 virtual PARTIDs. If MPAMIDR_EL1.VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, **MPAMVPM0_EL2**.

Virtual PARTID mapping is enabled by **MPAMHCR_EL2.EL1_VPMEN** for PARTIDs in **MPAM1_EL1** and by **MPAMHCR_EL2.EL0_VPMEN** for **MPAM0_EL1**.

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is only valid when the **MPAMVPMV_EL2.VPM_V** bit in bit position n is set to 1.

**Configuration**

This register is present only when MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX == 111. Otherwise, direct accesses to MPAMVPM7_EL2 are **UNDEFINED**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAMVPM7_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPM7_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| PhyPARTID31 | PhyPARTID29 | PhyPARTID30 | PhyPARTID28 |

**PhyPARTID31, bits [63:48]**

Virtual PARTID Mapping Entry for virtual PARTID 31. PhyPARTID31 gives the mapping of virtual PARTID 31 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID30, bits [47:32]**

Virtual PARTID Mapping Entry for virtual PARTID 30. PhyPARTID30 gives the mapping of virtual PARTID 30 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.
**PhyPARTID29, bits [31:16]**

Virtual PARTID Mapping Entry for virtual PARTID 29. PhyPARTID29 gives the mapping of virtual PARTID 29 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**PhyPARTID28, bits [15:0]**

Virtual PARTID Mapping Entry for virtual PARTID 28. PhyPARTID28 gives the mapping of virtual PARTID 28 to a physical PARTID.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPM7_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, MPAMVPM7_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x978];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      AArch64.SystemAccessTrap(EL2, 0x18);
    end else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return MPAMVPM7_EL2;
  elsif PSTATE.EL == EL3 then
    return MPAMVPM7_EL2;

MSR MPAMVPM7_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x978] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMVPM7_EL2 = X[t];
    end
elsif PSTATE.EL == EL3 then
    MPAMVPM7_EL2 = X[t];
MPAMVPMV_EL2, MPAM Virtual Partition Mapping Valid Register

The MPAMVPMV_EL2 characteristics are:

**Purpose**

Valid bits for virtual PARTID mapping entries. Each bit m corresponds to virtual PARTID mapping entry m in the MPAMVPM<n>_EL2 registers where n = m >> 2.

**Configuration**

This register is present only when MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPMV_EL2 are **UNDEFINED**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

MPAMVPMV_EL2 is a 64-bit register.

**Field descriptions**

The MPAMVPMV_EL2 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| VPM_V<m>, bit [m], for m = 0 to 31 |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**Bits [63:32]**

Reserved, RES0.

**VPM_V<m>, bit [m], for m = 0 to 31**

Contains valid bit for virtual PARTID mapping entry corresponding to virtual PARTID<m>.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the MPAMVPMV_EL2**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x938];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
    end
elif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPMV_EL2;
    end
elsif PSTATE.EL == EL3 then
    MPAMVPMV_EL2 = X[t];

MSR MPAMVPMV_EL2, <Xt>
The MPIDR_EL1 characteristics are:

**Purpose**

In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

**Configuration**

AArch64 System register MPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MPIDR[31:0].

In a uniprocessor system Arm recommends that each Aff<n> field of this register returns a value of 0.

**Attributes**

MPIDR_EL1 is a 64-bit register.

**Field descriptions**

The MPIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [63:40]**

Reserved, RES0.

**Aff3, bits [39:32]**

Affinity level 3. See the description of Aff0 for more information.

Aff3 is not supported in AArch32 state.

**Bit [31]**

Reserved, RES1.

**U, bit [30]**

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system. The possible values of this bit are:

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Processor is part of a multiprocessor system.</td>
</tr>
<tr>
<td>0b1</td>
<td>Processor is part of a uniprocessor system.</td>
</tr>
</tbody>
</table>

**Bits [29:25]**

Reserved, RES0.
MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels. The possible values of this bit are:

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Performance of PEs at the lowest affinity level, or PEs with MPIDR_EL1.MT set to 1, different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.</td>
</tr>
<tr>
<td>0b1</td>
<td>Performance of PEs at the lowest affinity level, or PEs with MPIDR_EL1.MT set to 1, different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.</td>
</tr>
</tbody>
</table>

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

Aff0, bits [7:0]

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

Accessing the MPIDR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, MPIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b0101</td>
<td></td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') & & HFGRTR_EL2.MPIDR_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() & & !ELUsingAArch32(EL2) then
      return VMPIDR_EL2;
    else
      return MPIDR_EL1;
  elsif PSTATE.EL == EL2 then
    return MPIDR_EL1;
  elsif PSTATE.EL == EL3 then
    return MPIDR_EL1;
The MVFR0_EL1 characteristics are:

**Purpose**

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with MVFR1_EL1 and MVFR2_EL1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register MVFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR0[31:0].

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

In an AArch64 only implementation, this register is **UNKNOWN**.

**Attributes**

MVFR0_EL1 is a 64-bit register.

**Field descriptions**

The MVFR0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>FPRound</td>
<td>Floating-Point Rounding modes</td>
</tr>
<tr>
<td>61</td>
<td>FPShrVec</td>
<td>Floating-Point Shl/Vec</td>
</tr>
<tr>
<td>60</td>
<td>FPSqrt</td>
<td>Floating-Point Sqrt</td>
</tr>
<tr>
<td>59</td>
<td>FPDivide</td>
<td>Floating-Point Divide</td>
</tr>
<tr>
<td>58</td>
<td>FPTrap</td>
<td>Floating-Point Trap</td>
</tr>
<tr>
<td>57</td>
<td>FPDP</td>
<td>Floating-Point DP</td>
</tr>
<tr>
<td>56</td>
<td>FPSP</td>
<td>Floating-Point SP</td>
</tr>
<tr>
<td>55</td>
<td>SIMDReg</td>
<td>SIMD Register</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**FPRound, bits [31:28]**

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes. Defined values are:

<table>
<thead>
<tr>
<th>FPRound</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the <strong>FPSCR</strong> setting.</td>
</tr>
<tr>
<td>0b0001</td>
<td>All rounding modes supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.
FPShVec, bits [27:24]

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors. Defined values are:

<table>
<thead>
<tr>
<th>FPShVec</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Short vectors not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Short vector operation supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

FPSqrt, bits [23:20]

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations. Defined values are:

<table>
<thead>
<tr>
<th>FPSqrt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

FPDivide, bits [19:16]

Indicates whether the floating-point implementation provides support for VFP divide operations. Defined values are:

<table>
<thead>
<tr>
<th>FPDivide</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

FPTrap, bits [15:12]

Floating Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping. Defined values are:

<table>
<thead>
<tr>
<th>FPTrap</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

A value of 0b0001 indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

FPDP, bits [11:8]

Double Precision. Indicates whether the floating-point implementation provides support for double-precision operations. Defined values are:
FPDP, bits [7:4]

Single Precision. Indicates whether the floating-point implementation provides support for single-precision operations. Defined values are:

<table>
<thead>
<tr>
<th>FPDP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported, VFPv2.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is 0b0001.
- VDIV.F64 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

FPSP, bits [7:4]

Single Precision. Indicates whether the floating-point implementation provides support for single-precision operations. Defined values are:

<table>
<thead>
<tr>
<th>FPSP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported, VFPv2.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

SIMDReg, bits [3:0]

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank. Defined values are:

<table>
<thead>
<tr>
<th>SIMDReg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The implementation has no Advanced SIMD and floating-point support.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation includes floating-point support with 16 x 64-bit registers.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

Accessing the MVFR0_EL1

Accesses to this register use the following encodings:
MRS <Xt>, MVFR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      return MVFR0_EL1;
  elsif PSTATE.EL == EL2 then
    return MVFR0_EL1;
  elsif PSTATE.EL == EL3 then
    return MVFR0_EL1;
The MVFR1_EL1 characteristics are:

**Purpose**

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with MVFR0_EL1 and MVFR2_EL1.

For general information about the interpretation of the ID registers see ‘Principles of the ID scheme for fields in ID registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

AArch64 System register MVFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR1[31:0].

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

In an AArch64 only implementation, this register is UNKNOWN.

**Attributes**

MVFR1_EL1 is a 64-bit register.

**Field descriptions**

The MVFR1_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:32]</th>
<th>Reserved, RES0</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIMDFMAC</td>
<td>FPHP</td>
</tr>
<tr>
<td>SIMDHP</td>
<td>SIMDSP</td>
</tr>
<tr>
<td>SIMDInt</td>
<td>SIMDLS</td>
</tr>
<tr>
<td>FPDNaN</td>
<td>FPFTZ</td>
</tr>
</tbody>
</table>

**SIMDFMAC, bits [31:28]**

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDFMAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.
**FPHP, bits [27:24]**

Floating Point Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

<table>
<thead>
<tr>
<th>FPHP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds instructions for conversion between double-precision and half-precision.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and adds support for half-precision floating-point arithmetic.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 in an implementation without floating-point support.
- 0b0010 in an implementation with floating-point support that does not include the ARMv8.2-FP16 extension.
- 0b0011 in an implementation with floating-point support that includes the ARMv8.2-FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

<table>
<thead>
<tr>
<th>Half Precision instructions supported</th>
<th>FPHP</th>
<th>SIMDHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>No support</td>
<td>0b0000</td>
<td>0b0000</td>
</tr>
<tr>
<td>Conversions only</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
<tr>
<td>Conversions and arithmetic</td>
<td>0b0011</td>
<td>0b0010</td>
</tr>
</tbody>
</table>

**SIMDHP, bits [23:20]**

Advanced SIMD Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

<table>
<thead>
<tr>
<th>SIMDHP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds support for half-precision floating-point arithmetic.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 in an implementation without SIMD floating-point support.
- 0b0010 in an implementation with SIMD floating-point support that does not include the ARMv8.2-FP16 extension.
- 0b0011 in an implementation with SIMD floating-point support that includes the ARMv8.2-FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

<table>
<thead>
<tr>
<th>Half Precision instructions supported</th>
<th>FPHP</th>
<th>SIMDHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>No support</td>
<td>0b0000</td>
<td>0b0000</td>
</tr>
<tr>
<td>Conversions only</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
<tr>
<td>Conversions and arithmetic</td>
<td>0b0011</td>
<td>0b0010</td>
</tr>
</tbody>
</table>

**SIMDSP, bits [19:16]**

Advanced SIMD Single Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDSP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented. This value is permitted only if the SIMDiInt field is 0b0001.</td>
</tr>
</tbody>
</table>
All other values are reserved.

In Armv8-A, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

**SIMDInt, bits [15:12]**

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDInt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0000}</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>\texttt{0b0001}</td>
<td>Implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

**SIMDLS, bits [11:8]**

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDLS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0000}</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>\texttt{0b0001}</td>
<td>Implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

**FPDNaN, bits [7:4]**

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode. Defined values are:

<table>
<thead>
<tr>
<th>FPDNaN</th>
<th>Meaning</th>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0000}</td>
<td>Not implemented, or hardware supports only the Default NaN mode.</td>
<td></td>
</tr>
<tr>
<td>\texttt{0b0001}</td>
<td>Hardware supports propagation of NaN values.</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

**FPFtZ, bits [3:0]**

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation. Defined values are:

<table>
<thead>
<tr>
<th>FPFtZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0000}</td>
<td>Not implemented, or hardware supports only the Flush-to-Zero mode of operation.</td>
</tr>
<tr>
<td>\texttt{0b0001}</td>
<td>Hardware supports full denormalized number arithmetic.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are \texttt{0b0000} and \texttt{0b0001}.

**Accessing the MVFR1_EL1**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    else
        UNDEFINED;
    end if
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MVFR1_EL1;
    end if
elsif PSTATE.EL == EL2 then
    return MVFR1_EL1;
elsif PSTATE.EL == EL3 then
    return MVFR1_EL1;
The MVFR2_EL1 characteristics are:

**Purpose**

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with MVFR0_EL1 and MVFR1_EL1.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D7.1.3.

**Configuration**

AArch64 System register MVFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register MVFR2[31:0].

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

In an AArch64 only implementation, this register is UNKNOWN.

**Attributes**

MVFR2_EL1 is a 64-bit register.

**Field descriptions**

The MVFR2_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | FPMisc | SIMDMisc |

**Bits [63:8]**

Reserved, RES0.

**FPMisc, bits [7:4]**

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

<table>
<thead>
<tr>
<th>FPMisc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or no support for miscellaneous features.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Floating-point selection.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As 0b0010, and Floating-point Round to Integer Floating-point.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As 0b0011, and Floating-point MaxNum and MinNum.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0100.
**SIMDMisc, bits [3:0]**

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.

<table>
<thead>
<tr>
<th>SIMDMisc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or no support for miscellaneous features.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Floating-point Conversion to Integer with Directed Rounding modes.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, and Floating-point Round to Integer Floating-point.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As 0b0010, and Floating-point MaxNum and MinNum.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0011.

**Accessing the MVFR2_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, MVFR2_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return MVFR2_EL1;
elsif PSTATE.EL == EL2 then
  return MVFR2_EL1;
elsif PSTATE.EL == EL3 then
  return MVFR2_EL1;
```
NZCV, Condition Flags

The NZCV characteristics are:

**Purpose**

Allows access to the condition flags.

**Configuration**

**Attributes**

NZCV is a 64-bit register.

**Field descriptions**

The NZCV bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>Reserved</td>
</tr>
<tr>
<td>61</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>Reserved</td>
</tr>
<tr>
<td>55</td>
<td>Reserved</td>
</tr>
<tr>
<td>54</td>
<td>Reserved</td>
</tr>
<tr>
<td>53</td>
<td>Reserved</td>
</tr>
<tr>
<td>52</td>
<td>Reserved</td>
</tr>
<tr>
<td>51</td>
<td>Reserved</td>
</tr>
<tr>
<td>50</td>
<td>Reserved</td>
</tr>
<tr>
<td>49</td>
<td>Reserved</td>
</tr>
<tr>
<td>48</td>
<td>Reserved</td>
</tr>
<tr>
<td>47</td>
<td>Reserved</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
</tr>
<tr>
<td>31</td>
<td>N, bit</td>
</tr>
<tr>
<td>30</td>
<td>Z, bit</td>
</tr>
<tr>
<td>29</td>
<td>C, bit</td>
</tr>
<tr>
<td>28</td>
<td>V, bit</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative condition flag. Set to 1 if the result of the last flag-setting instruction was negative.

**Z, bit [30]**

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

**C, bit [29]**

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

**V, bit [28]**

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

**Bits [27:0]**

Reserved, RES0.

**Accessing the NZCV**

Accesses to this register use the following encodings:
MRS <Xt>, NZCV

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL1 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL2 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL3 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);

MSR NZCV, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
elsif PSTATE.EL == EL1 then
elsif PSTATE.EL == EL2 then
elsif PSTATE.EL == EL3 then

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
OSDLR_EL1, OS Double Lock Register

The OSDLR_EL1 characteristics are:

**Purpose**

Used to control the OS Double Lock.

**Configuration**

AArch64 System register OSDLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSDLR[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

OSDLR_EL1 is a 64-bit register.

**Field descriptions**

The OSDLR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>Reserved</td>
</tr>
<tr>
<td>61</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>Reserved</td>
</tr>
<tr>
<td>55</td>
<td>Reserved</td>
</tr>
<tr>
<td>54</td>
<td>Reserved</td>
</tr>
<tr>
<td>53</td>
<td>Reserved</td>
</tr>
<tr>
<td>52</td>
<td>Reserved</td>
</tr>
<tr>
<td>51</td>
<td>Reserved</td>
</tr>
<tr>
<td>50</td>
<td>Reserved</td>
</tr>
<tr>
<td>49</td>
<td>Reserved</td>
</tr>
<tr>
<td>48</td>
<td>Reserved</td>
</tr>
<tr>
<td>47</td>
<td>Reserved</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:1]**

Reserved, RES0.

**DLK, bit [0]**

*When ARMv8.0-DoubleLock is implemented:*

OS Double Lock control bit.

<table>
<thead>
<tr>
<th>DLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>OS Double Lock unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>OS Double Lock locked, if DBGPRCR_EL1.CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**Otherwise:**

Reserved, RAZ/WI.

**Accessing the OSDLR_EL1**

Accesses to this register use the following encodings:
MRS <Xt>, OSDLR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
  IsFeatureImplemented("ARMv8.0-DoubleLock") && HDFGRTR_EL2.OSDLR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elif EL2Enabled() && !ELUsingAArch32(EL2) && CRn != '00' &&
  (IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by
  MDCR_EL2.TDOSA") then
    AArch64.SystemAccessTrap(EL2, 0x18);
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && CRn == '1' &&
  IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by
  MDCR_EL3.TDOSA") then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return OSDLR_EL1;
elif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CRn == '1' &&
  (IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by
  MDCR_EL3.TDOSA") then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  OSDLR_EL1 = X[t];
elif PSTATE.EL == EL3 then
  OSDLR_EL1 = X[t];
else
  return OSDLR_EL1;
OsdtrrX_EL1, OS lock Data Transfer Register, receive

The OSDTRRX_EL1 characteristics are:

**Purpose**

Used for save/restore of DBGDTRRX_EL0. It is a component of the Debug Communications Channel.

**Configuration**

AArch64 System register OSDTRRX_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXext[31:0].

**Attributes**

OSDTRRX_EL1 is a 64-bit register.

**Field descriptions**

The OSDTRRX_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, res0</td>
</tr>
<tr>
<td>31</td>
<td>Update DTRRX without side-effect</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, res0.

**Bits [31:0]**

Update DTRRX without side-effect.

Writes to this register update the value in DTRRX and do not change RXfull.

Reads of this register return the last value written to DTRRX and do not change RXfull.

For the full behavior of the Debug Communications Channel, see ‘The Debug Communication Channel and Instruction Transfer Register’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter H4.

**Accessing the OSDTRRX_EL1**

Arm deprecates reads and writes of OSDTRRX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings:

MRS <Xt>, OSDTRRX_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return OSDTRRX_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      OSDTRRX_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    return OSDTRRX_EL1;

MSR OSDTRRX_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSDTRRX_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      OSDTRRX_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    OSDTRRX_EL1 = X[t];
OSDTRTX_EL1, OS Lock Data Transfer Register, Transmit

The OSDTRTX_EL1 characteristics are:

**Purpose**

Used for save/restore of DBGDTRTX_EL0. It is a component of the Debug Communications Channel.

**Configuration**

AArch64 System register OSDTRTX_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXext[31:0].

**Attributes**

OSDTRTX_EL1 is a 64-bit register.

**Field descriptions**

The OSDTRTX_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Return DTRTX without side-effect</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Return DTRTX without side-effect</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**Bits [31:0]**

Return DTRTX without side-effect.

Reads of this register return the value in DTRTX and do not change TXfull.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter H4.

**Accessing the OSDTRTX_EL1**

Arm deprecates reads and writes of OSDTRTX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings:

```
MRS <Xt>, OSDTRTX_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return OSDTRTX_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSDTRTX_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  OSDTRTX_EL1 = X[t];

MSR OSDTRTX_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSDTRTX_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSDTRTX_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  OSDTRTX_EL1 = X[t];
OSECCR_EL1, OS Lock Exception Catch Control Register

The OSECCR_EL1 characteristics are:

**Purpose**

Provides a mechanism for an operating system to access the contents of EDECCR that are otherwise invisible to software, so it can save/restore the contents of EDECCR over powerdown on behalf of the external debugger.

**Configuration**

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOOSECCR[31:0].

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to External register EDECCR[31:0].

If OSLSR_EL1.OSLK == 0, then OSECCR_EL1 returns an UNKNOWN value on reads and ignores writes.

**Attributes**

OSECCR_EL1 is a 64-bit register.

**Field descriptions**

The OSECCR_EL1 bit assignments are:

**When DBGOSLSR.OSLK == 1:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>31</td>
<td>EDECCR</td>
<td>Used for save/restore to EDECCR over powerdown.</td>
</tr>
</tbody>
</table>

Bits [63:32]:

Reserved, RES0.

**EDECCR, bits [31:0]**

Used for save/restore to EDECCR over powerdown.

Reads or writes to this field are indirect accesses to EDECCR.

**Accessing the OSECCR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, OSECCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.OSECCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return OSECCR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return OSECCR_EL1;
elsif PSTATE.EL == EL3 then
  return OSECCR_EL1;

MSR OSECCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.OSECCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSECCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSECCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  OSECCR_EL1 = X[t];
OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

**Purpose**

Used to lock or unlock the OS Lock.

**Configuration**

AArch64 System register OSLAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSLAR[31:0].

AArch64 System register OSLAR_EL1 bits [31:0] are architecturally mapped to External register OSLAR_EL1[31:0].

**Attributes**

OSLAR_EL1 is a 64-bit register.

**Field descriptions**

The OSLAR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>59</td>
<td>OSLK</td>
</tr>
<tr>
<td>31</td>
<td>OSLK</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:1]**

Reserved, RES0.

**OSLK, bit [0]**

On writes to OSLAR_EL1, bit[0] is copied to the OS Lock.

Use OSLSR_EL1.OSLK to check the current status of the lock.

**Accessing the OSLAR_EL1**

Accesses to this register use the following encodings:

MSR OSLAR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HDFGWTR_EL2.OSLAR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSLAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    OSLAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  OSLAR_EL1 = X[t];
OSLSR_EL1, OS Lock Status Register

The OSLSR_EL1 characteristics are:

**Purpose**

Provides the status of the OS Lock.

**Configuration**

AArch64 System register OSLSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSLSR[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

OSLSR_EL1 is a 64-bit register.

**Field descriptions**

The OSLSR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>nTT</td>
</tr>
<tr>
<td>1</td>
<td>OSLM[1]</td>
</tr>
<tr>
<td>0</td>
<td>OSLM[0]</td>
</tr>
</tbody>
</table>

**Bits [63:4]**

Reserved, RES0.

**OSLM[1], bit [3]**

This field is bit[1] of OSLM[1:0].

OS lock model implemented. Identifies the form of OS save and restore mechanism implemented.

<table>
<thead>
<tr>
<th>OSLM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>OS Lock not implemented.</td>
</tr>
<tr>
<td>10</td>
<td>OS Lock implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

The OSLM field is split as follows:

- OSLM[1] is OSLSR_EL1[3].
- OSLM[0] is OSLSR_EL1[0].

**nTT, bit [2]**

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.
OSLK, bit [1]

OS Lock Status.

<table>
<thead>
<tr>
<th>OSLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>OS Lock unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>OS Lock locked.</td>
</tr>
</tbody>
</table>

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The following resets apply:

- On a Cold reset, this field resets to 1.
- On a Warm reset, the value of this field is unchanged.

OSLM[0], bit [0]

This field is bit[0] of OSLM[1:0].

See OSLM[1] for the field description.

Accessing the OSLSR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, OSLSR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b10</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.OSLR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return OSLSR_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return OSLSR_EL3;
  elsif PSTATE.EL == EL3 then
    return OSLSR_EL1;
```
PAN, Privileged Access Never

The PAN characteristics are:

**Purpose**

Allows access to the Privileged Access Never bit.

**Configuration**

This register is present only when ARMv8.1-PAN is implemented. Otherwise, direct accesses to PAN are **UNDEFINED**.

**Attributes**

PAN is a 64-bit register.

**Field descriptions**

The PAN bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 |  RES0 | PAN | RES0 | RES0 |

**Bits [63:23]**

Reserved, RES0.

**PAN, bit [22]**

Privileged Access Never.

<table>
<thead>
<tr>
<th>PAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Privileged reads and write are not disabled by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>Disables privileged read and write accesses to addresses accessible at EL0 for an enabled stage 1 translation regime that defines the EL0 permissions.</td>
</tr>
</tbody>
</table>

The value of this bit is usually preserved on taking an exception, except in the following situations:

- When the target of the exception is EL1, and the value of the SCTLR_EL1.SPAN bit is 0, this bit is set to 1.
- When the target of the exception is EL2, HCR_EL2.{E2H, TGE} is {1, 1}, and the value of the SCTLR_EL2.SPAN bit is 0, this bit is set to 1.

**Bits [21:0]**

Reserved, RES0.

**Accessing the PAN**

For details on the operation of the MSR (immediate) accessor, see MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:
MRS `<Xt>`, PAN

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  return Zeros(41):PSTATE.PAN:Zeros(22);
elsif PSTATE.EL == EL2 then
  return Zeros(41):PSTATE.PAN:Zeros(22);
elsif PSTATE.EL == EL3 then
  return Zeros(41):PSTATE.PAN:Zeros(22);

MSR PAN, `<Xt>`

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  PSTATE.PAN = X[t]<22 >;
elsif PSTATE.EL == EL2 then
  PSTATE.PAN = X[t]<22 >;
elsif PSTATE.EL == EL3 then
  PSTATE.PAN = X[t]<22 >;

MSR PAN, `<imm>`

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
PAR_EL1, Physical Address Register

The PAR_EL1 characteristics are:

**Purpose**

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

**Configuration**

AArch64 System register PAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register PAR[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PAR_EL1 is a 64-bit register.

**Field descriptions**

The PAR_EL1 bit assignments are:

**When PAR_EL1.F == 0b0:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATTR</strong></td>
<td><strong>RES0</strong></td>
<td><strong>PA[51:48]</strong></td>
<td><strong>PA[47:12]</strong></td>
<td><strong>RES1</strong></td>
<td><strong>IMPLEMENTATION DEFINED</strong></td>
<td><strong>NS</strong></td>
<td><strong>SH</strong></td>
<td><strong>RES0</strong></td>
<td><strong>F</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

**ATTR, bits [63:56]**

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR_EL1, MAIR_EL2, and MAIR_EL3.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally UNKNOWN value.

**Bits [55:52]**

Reserved, RES0.

**PA[51:48], bits [51:48]**
When ARMv8.2-LPA is implemented:

Extension to PA[47:12]. See PA[47:12] for more details.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[47:12], bits [47:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[47:12].

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, the PA[51:48] bits form the upper part of the address value. Otherwise the PA[51:48] bits are RES0.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

This field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

NS, bit [9]

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when SCR_EL3.EEL2 is 1, this bit reflects the Security state of the intermediate physical address space of the translation for the instructions:

- In AArch64 state: AT S1E1R, AT S1E1W, AT S1E1RP, AT S1E1WP, AT S1E0R, and AT S1E0W.
- In AArch32 state: ATS1CPR, ATS1CPW, ATS1CPRP, ATS1CPWP, ATS1CUR, and ATS1CUW.

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address. Permitted values are:

<table>
<thead>
<tr>
<th>SH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

The value 0b01 is reserved.

Note
This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally **UNKNOWN** value.

**Bits [6:1]**

Reserved, RES0.

**F, bit [0]**

Indicates whether the instruction performed a successful address translation.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Address translation completed successfully.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**When PAR_EL1.F == 0b1:**

<table>
<thead>
<tr>
<th>63  62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</th>
<th>IMPLEMENTATION DEFINED</th>
<th>IMPLEMENTATION DEFINED</th>
<th>IMPLEMENTATION DEFINED</th>
<th>RES0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>RES0</td>
<td>RES1</td>
<td>RES0</td>
<td>S</td>
</tr>
</tbody>
</table>

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

**IMPLEMENTATION DEFINED, bits [63:56]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally **UNKNOWN** value.

**IMPLEMENTATION DEFINED, bits [55:52]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally **UNKNOWN** value.

**IMPLEMENTATION DEFINED, bits [51:48]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally **UNKNOWN** value.

**Bits [47:12]**

Reserved, RES0.

**Bit [11]**

Reserved, RES1.
Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

<table>
<thead>
<tr>
<th>S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Translation aborted because of a fault in the stage 1 translation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Translation aborted because of a fault in the stage 2 translation.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

This field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort ESR encoding.

This field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Address translation aborted.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Accessing the PAR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PAR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return PAR_EL1;
    elsif PSTATE.EL == EL2 then
        return PAR_EL1;
    elsif PSTATE.EL == EL3 then
        return PAR_EL1;
endif

MSR PAR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        PAR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        PAR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PAR_EL1 = X[t];
endif
PMBIDR_EL1, Profiling Buffer ID Register

The PMBIDR_EL1 characteristics are:

**Purpose**

Provides information to software as to whether the buffer can be programmed at the current Exception level.

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBIDR_EL1 are **UNDEFINED**.

**Attributes**

PMBIDR_EL1 is a 64-bit register.

**Field descriptions**

The PMBIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-6</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
<td>Flag updates. Defines whether the address translation performed by the Profiling Buffer manages the Access Flag and dirty state.</td>
</tr>
<tr>
<td>4</td>
<td>P</td>
<td>Programming not allowed. The Profiling Buffer is owned by a higher Exception level or the other Security state.</td>
</tr>
</tbody>
</table>

**Bits [63:6]**

Reserved, RES0.

**F, bit [5]**

Flag updates. Defines whether the address translation performed by the Profiling Buffer manages the Access Flag and dirty state.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hardware management of the Access Flag and dirty state for accesses made by the Statistical Profiling Extension is always disabled for all translation stages.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hardware management for the Access Flag and dirty state for accesses made by the Statistical Profiling Extension is controlled in the same way as explicit memory accesses in the owning translation regime.</td>
</tr>
</tbody>
</table>

If hardware management of the Access Flag is disabled for a stage of translation, an access to Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.

If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

**P, bit [4]**

Programming not allowed. The Profiling Buffer is owned by a higher Exception level or the other Security state.
The value read from this field depends on the current Exception level and the Effective values of `MDCR_EL3.NSPB` and `MDCR_EL2.E2PB`:

- If EL3 is implemented, and either `MDCR_EL3.NSPB == 0b00` or `MDCR_EL3.NSPB == 0b01`, this bit reads as one from:
  - Non-secure EL1.
  - Non-secure EL2.
  - If Secure EL2 is implemented and enabled, and `MDCR_EL2.E2PB == 0b00`, Secure EL1.

- If EL3 is implemented, and either `MDCR_EL3.NSPB == 0b10` or `MDCR_EL3.NSPB == 0b11`, this bit reads as one from:
  - Secure EL1.
  - If Secure EL2 is implemented, Secure EL2.
  - If EL2 is implemented and `MDCR_EL2.E2PB == 0b00`, Non-secure EL1.
  - If EL3 is not implemented, EL2 is implemented, and `MDCR_EL2.E2PB == 0b00`, this bit reads as one from EL1.
  - Otherwise, this bit reads as zero.

### Align, bits [3:0]

Defines the minimum alignment constraint for `PMBPTR_EL1`. If this field is non-zero, then the PE must pad every record up to a multiple of this size.

<table>
<thead>
<tr>
<th>Align</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Byte</td>
</tr>
<tr>
<td>0b0001</td>
<td>Halfword.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Word.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Doubleword.</td>
</tr>
<tr>
<td>0b0100</td>
<td>16 Bytes.</td>
</tr>
<tr>
<td>0b0101</td>
<td>32 Bytes.</td>
</tr>
<tr>
<td>0b0110</td>
<td>64 Bytes.</td>
</tr>
<tr>
<td>0b0111</td>
<td>128 Bytes.</td>
</tr>
<tr>
<td>0b1000</td>
<td>256 Bytes.</td>
</tr>
<tr>
<td>0b1001</td>
<td>512 Bytes.</td>
</tr>
<tr>
<td>0b1010</td>
<td>1KB.</td>
</tr>
<tr>
<td>0b1011</td>
<td>2KB.</td>
</tr>
</tbody>
</table>

For more information, see Restrictions on the current write pointer.

### Accessing the PMBIDR_EL1

Accesses to this register use the following encodings:

```assembly
MRS <Xt>, PMBIDR_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b111</td>
</tr>
</tbody>
</table>

```assembly
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  return PMBIDR_EL1;
elsif PSTATE.EL == EL2 then
  return PMBIDR_EL1;
elsif PSTATE.EL == EL3 then
  return PMBIDR_EL1;
```
The PMBLIMITR_EL1 characteristics are:

**Purpose**

Defines the upper limit for the profiling buffer, and enables the profiling buffer

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBLIMITR_EL1 are undefined.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally unknown values.

**Attributes**

PMBLIMITR_EL1 is a 64-bit register.

**Field descriptions**

The PMBLIMITR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:12</td>
<td>LIMIT</td>
<td>Limit address. PMBLIMITR_EL1.LIMIT:Zeros(12) is the address of the first byte in memory after the last byte in the profiling buffer. If the smallest implemented translation granule is not 4KB, then bits[N-1:12] are RES0, where N is the implementation defined value, Log2(smallest implemented translation granule). This field resets to an architecturally unknown value.</td>
</tr>
<tr>
<td>11:3</td>
<td>Reserved, RES0.</td>
<td></td>
</tr>
<tr>
<td>2:1</td>
<td>Fill mode</td>
<td></td>
</tr>
</tbody>
</table>

**LIMIT, bits [63:12]**

Limit address. PMBLIMITR_EL1.LIMIT:Zeros(12) is the address of the first byte in memory after the last byte in the profiling buffer. If the smallest implemented translation granule is not 4KB, then bits[N-1:12] are RES0, where N is the implementation defined value, Log2(smallest implemented translation granule).

This field resets to an architecturally unknown value.

**Bits [11:3]**

Reserved, RES0.

**FM, bits [2:1]**

Fill mode

<table>
<thead>
<tr>
<th>FM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Stop collection and raise maintenance interrupt on buffer fill.</td>
</tr>
</tbody>
</table>

All other values are reserved. If this field is programmed with a reserved value, the PE behaves as if this field has a defined value, other than for a direct read of the register. Software must not rely on the behavior of reserved values, as they might change in a future version of the architecture.

This field resets to an architecturally unknown value.
E, bit [0]

Profiling Buffer enable

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All output is discarded.</td>
</tr>
<tr>
<td>0b1</td>
<td>Profiling buffer enabled.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**Accessing the PMBLIMITR_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, PMBLIMITR_EL1**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMBLIMITR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    return NVMem[0x800];
  else
    return PMBLIMITR_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMBLIMITR_EL1;
  elsif PSTATE.EL == EL3 then
    return PMBLIMITR_EL1;
```

**MSR PMBLIMITR_EL1, <Xt>**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.PMBLIMITR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    NVMem[0x800] = X[t];
  else
    PMBLIMITR_EL1 = X[t];
  end if;
else
  PMBLIMITR_EL1 = X[t];
end if;
if PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMBLIMITR_EL1 = X[t];
  end if;
else
  PMBLIMITR_EL1 = X[t];
end if;
PMBPTR_EL1, Profiling Buffer Write Pointer Register

The PMBPTR_EL1 characteristics are:

**Purpose**

Defines the current write pointer for the profiling buffer.

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBPTR_EL1 are **UNDEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

PMBPTR_EL1 is a 64-bit register.

**Field descriptions**

The PMBPTR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td><strong>PTR</strong></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>61</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>59</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>57</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>47</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td><strong>PTR</strong></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

**PTR, bits [63:0]**

Current write address. Defines the virtual address of the next entry to be written to the buffer.

The architecture places restrictions on the values software can write to the pointer. For more information see 'Restrictions on the current write pointer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D6.3.5.

**Note**

As a result, an implementation might treat some of bits[M:0], where M is defined by PMBIDR_EL1.Align, as **RES0**.

On a management interrupt, PMBPTR_EL1 is frozen.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMBPTR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, PMBPTR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
MSR PMBPTR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.PMBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
        MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
        MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x810];
    else
        return PMBPTR_EL1;
else
    PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) &&
            SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) &&
            SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMBPTR_EL1;
        else
            return PMBPTR_EL1;
    else
        PSTATE.EL == EL3 then
            return PMBPTR_EL1;

    if PSTATE.EL == EL0 then
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
        SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.PMBPTR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) &&
        SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) &&
        SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x810];
    else
        return PMBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) &&
        SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) &&
        SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMBPTR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMBPTR_EL1 = X[t];
PMBSR_EL1, Profiling Buffer Status/syndrome Register

The PMBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software when the buffer is disabled because the management interrupt has been raised.

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMBSR_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMBSR_EL1 is a 64-bit register.

Field descriptions

The PMBSR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| EC | RES0 | DL | EA | S | COLL | MSS |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EC, bits [31:26]

Exception class

Top-level description of the cause of the buffer management event

<table>
<thead>
<tr>
<th>EC</th>
<th>Meaning</th>
<th>MSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Other buffer management event. All buffer management events other than</td>
<td>MSS encoding for other buffer management</td>
</tr>
<tr>
<td></td>
<td>those described by other defined Exception class codes.</td>
<td>events</td>
</tr>
<tr>
<td>0b100100</td>
<td>Stage 1 Data Abort on write to Profiling Buffer.</td>
<td>MSS encoding for stage 1 or stage 2 Data Aborts on write to buffer</td>
</tr>
<tr>
<td>0b100101</td>
<td>Stage 2 Data Abort on write to Profiling Buffer.</td>
<td>MSS encoding for stage 1 or stage 2 Data Aborts on write to buffer</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
### Bits [25:20]

Reserved, RES0.

### DL, bit [19]

Partial record lost.

Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

<table>
<thead>
<tr>
<th>DL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer.</td>
</tr>
<tr>
<td>0b1</td>
<td>Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse. All records prior to the last record have been written to the buffer.</td>
</tr>
</tbody>
</table>

When the buffer management event was because of an asynchronous external abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### EA, bit [18]

External abort.

<table>
<thead>
<tr>
<th>EA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An external abort has not been asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>An external abort has been asserted and detected by the Statistical Profiling Extension.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the PE never sets this bit as the result of an External abort.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### S, bit [17]

Service

<table>
<thead>
<tr>
<th>S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PMBIRQ is not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>PMBIRQ is asserted. All profiling data has either been written to the buffer or discarded.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### COLL, bit [16]

Collision detected.

<table>
<thead>
<tr>
<th>COLL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No collision events detected.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one collision event was recorded.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.
MSS, bits [15:0]

Management Event Specific Syndrome.

Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### MSS encoding for stage 1 or stage 2 Data Aborts on write to buffer

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>FSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [15:6]**

Reserved, RES0.

**FSC, bits [5:0]**

Fault status code

<table>
<thead>
<tr>
<th>FSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000xx</td>
<td>Address Size fault. Bits [1:0] encode the level.</td>
</tr>
<tr>
<td>0b0001xx</td>
<td>Translation fault. Bits [1:0] encode the level.</td>
</tr>
<tr>
<td>0b0010xx</td>
<td>Access Flag fault. Bits [1:0] encode the level.</td>
</tr>
<tr>
<td>0b0011xx</td>
<td>Permission fault. Bits [1:0] encode the level.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort on write.</td>
</tr>
<tr>
<td>0b0101xx</td>
<td>Synchronous External abort on translation table walk or hardware update of translation table. Bits [1:0] encode the level.</td>
</tr>
<tr>
<td>0b01001</td>
<td>Asynchronous External abort on write.</td>
</tr>
<tr>
<td>0b100001</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB Conflict fault.</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault. When ARMv8.1-TTHM is implemented</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see Faults and Watchpoints.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### MSS encoding for other buffer management events

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>BSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bits [15:6]
Reserved, RES0.

BSC, bits [5:0]
Buffer status code

<table>
<thead>
<tr>
<th>BSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Buffer not filled</td>
</tr>
<tr>
<td>0b000001</td>
<td>Buffer filled</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMBSR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMBSR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.PMBSR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
    return PMBSR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    return NVMem[0x820];
  else
    return PMBSR_EL1;
elsif PSTATE.EL == EL3 then
  return PMBSR_EL1;

MSR PMBSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
  HDFGWTR_EL2.PMBSR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2PB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMBSR_EL1 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMBSR_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  PMBSR_EL1 = X[t];
PMCCFILTR_EL0, Performance Monitors Cycle Count Filter Register

The PMCCFILTR_EL0 characteristics are:

**Purpose**
Determines the modes in which the Cycle Counter, PMCCNTR_EL0, increments.

**Configuration**
AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCCFILTR[31:0].

AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to External register PMCCFILTR_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCCFILTR_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**
PMCCFILTR_EL0 is a 64-bit register.

**Field descriptions**
The PMCCFILTR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U</td>
<td>NSK</td>
<td>NSU</td>
<td>NSH</td>
<td>M</td>
<td>RES</td>
<td>SH</td>
<td>T</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
<td>RES</td>
</tr>
</tbody>
</table>

Reserved, RES0.

**P, bit [31]**
Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMCCFILTR_EL0.NSK bit.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count cycles in EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in EL1.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**U, bit [30]**
User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMCCFILTR_EL0.NSU bit.
PMCCFILTR_EL0, Performance Monitors Cycle Count Filter Register

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count cycles in EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**NSK, bit [29]**

**When HaveEL(EL3):**

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Non-secure EL1 are counted.

Otherwise, cycles in Non-secure EL1 are not counted.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**NSU, bit [28]**

**When HaveEL(EL3):**

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.U bit, cycles in Non-secure EL0 are counted.

Otherwise, cycles in Non-secure EL0 are not counted.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**NSH, bit [27]**

**When HaveEL(EL2):**

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If Secure EL2 is implemented, counting in Secure EL2 is further controlled by the PMCCFILTR_EL0.SH bit.

<table>
<thead>
<tr>
<th>NSH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not count cycles in EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count cycles in EL2.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**M, bit [26]**

**When HaveEL(EL3):**

Secure EL3 filtering bit.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Secure EL3 are counted.
Otherwise, cycles in Secure EL3 are not counted.

Most applications can ignore this field and set its value to 0.

---

**Note**

This field is not visible in the AArch32 PMCCFILTR System register.

---

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Bit [25]**

Reserved, RES0.

**SH, bit [24]**

*When ARMv8.4-SecEL2 is implemented:*

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMCCFILTR_EL0.NSH bit, cycles in Secure EL2 are counted.

Otherwise, cycles in Secure EL2 are not counted.

If Secure EL2 is not implemented or is disabled, this field is RES0.

---

**Note**

This field is not visible in the AArch32 PMCCFILTR System register.

---

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**T, bit [23]**

*When TME is implemented:*

Non-transactional state filtering bit.

<table>
<thead>
<tr>
<th>T</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit has no effect on filtering of cycles.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in Non-transactional state.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Bits [22:0]**

Reserved, RES0.
Accessing the PMCCFILTR_EL0

PMCCFILTR_EL0 can also be accessed by using PMXEVTYPER_EL0 with PMSELR_EL0 SEL set to 0b11111.

Accesses to this register use the following encodings:

MRS <Xt>, PMCCFILTR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b1111</td>
<td>0b11</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCCFILTR_EL0;
  endif
else if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCCFILTR_EL0;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCCFILTR_EL0;
  endif
elsif PSTATE.EL == EL3 then
  return PMCCFILTR_EL0;
endif

MSR PMCCFILTR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b1111</td>
<td>0b11</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            PMCCFILTR_EL0 = X[t];
        endif
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCCFILTR_EL0 = X[t];
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCFILTR_EL0 = X[t];
        endif
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCFILTR_EL0 = X[t];
        endif
    else
        PMCCFILTR_EL0 = X[t];
    endif
else
    PMCCFILTR_EL0 = X[t];
endif
PMCCNTR_EL0, Performance Monitors Cycle Count Register

The PMCCNTR_EL0 characteristics are:

**Purpose**

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for more information.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

**Configuration**

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0].

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCCNTR_EL0 are UNDEFINED.

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR_EL0 continues to increment when clocks are stopped by WFI and WFE instructions.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCCNTR_EL0 is a 64-bit register.

**Field descriptions**

The PMCCNTR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>CCNT</td>
</tr>
<tr>
<td>62</td>
<td>CCNT</td>
</tr>
<tr>
<td>61</td>
<td>CCNT</td>
</tr>
<tr>
<td>60</td>
<td>CCNT</td>
</tr>
<tr>
<td>59</td>
<td>CCNT</td>
</tr>
<tr>
<td>58</td>
<td>CCNT</td>
</tr>
<tr>
<td>57</td>
<td>CCNT</td>
</tr>
<tr>
<td>56</td>
<td>CCNT</td>
</tr>
<tr>
<td>55</td>
<td>CCNT</td>
</tr>
<tr>
<td>54</td>
<td>CCNT</td>
</tr>
<tr>
<td>53</td>
<td>CCNT</td>
</tr>
<tr>
<td>52</td>
<td>CCNT</td>
</tr>
<tr>
<td>51</td>
<td>CCNT</td>
</tr>
<tr>
<td>50</td>
<td>CCNT</td>
</tr>
<tr>
<td>49</td>
<td>CCNT</td>
</tr>
<tr>
<td>48</td>
<td>CCNT</td>
</tr>
<tr>
<td>47</td>
<td>CCNT</td>
</tr>
<tr>
<td>46</td>
<td>CCNT</td>
</tr>
<tr>
<td>45</td>
<td>CCNT</td>
</tr>
<tr>
<td>44</td>
<td>CCNT</td>
</tr>
<tr>
<td>43</td>
<td>CCNT</td>
</tr>
<tr>
<td>42</td>
<td>CCNT</td>
</tr>
<tr>
<td>41</td>
<td>CCNT</td>
</tr>
<tr>
<td>40</td>
<td>CCNT</td>
</tr>
<tr>
<td>39</td>
<td>CCNT</td>
</tr>
<tr>
<td>38</td>
<td>CCNT</td>
</tr>
<tr>
<td>37</td>
<td>CCNT</td>
</tr>
<tr>
<td>36</td>
<td>CCNT</td>
</tr>
<tr>
<td>35</td>
<td>CCNT</td>
</tr>
<tr>
<td>34</td>
<td>CCNT</td>
</tr>
<tr>
<td>33</td>
<td>CCNT</td>
</tr>
<tr>
<td>32</td>
<td>CCNT</td>
</tr>
</tbody>
</table>

**CCNT, bits [63:0]**

Cycle count. Depending on the values of PMCR_EL0.c{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMCCNTR_EL0**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2HTGE> != '1' && (!HaveEL(EL3) ||
        SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCCNTR_EL0;
        elseif PSTATE.EL == EL2 then
            if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return PMCCNTR_EL0;
        elseif PSTATE.EL == EL3 then
            return PMCCNTR_EL0;
MSR PMCCNTR_EL0, <Xt>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL2.FGTEn == '1' && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMCCNTR_EL0 = X[t];
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMCCNTR_EL0 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMCCNTR_EL0 = X[t];
  elsif PSTATE.EL == EL3 then
    PMCCNTR_EL0 = X[t];
PMCEID0_EL0, Performance Monitors Common Event Identification register 0

The PMCEID0_EL0 characteristics are:

### Purpose

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the ranges 0x0000 to 0x001F and 0x4000 to 0x401F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

---

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

---

For more information about the common events and the use of the PMCEID<n>_EL0 registers see the section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

### Configuration

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID0[31:0].

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to AArch32 System register PMCEID2[31:0].

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to External register PMCEID0[31:0].

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to External register PMCEID2[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCEID0_EL0 are UNDEFINED.

### Attributes

PMCEID0_EL0 is a 64-bit register.

### Field descriptions

The PMCEID0_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDhi&lt;n&gt;, bit [n+32], for n = 0 to 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When ARMv8.1-PMU is implemented:

IDhi[n] corresponds to common event (0x4000 + n).

For each bit:
A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

**Otherwise:**

Reserved, RES0.

**ID<n>, bit [n], for n = 0 to 31**

ID[n] corresponds to common event n.

For each bit:

<table>
<thead>
<tr>
<th>ID&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

**Accessing the PMCEID0_EL0**

Accesses to this register use the following encodings:

**MRS <Xt>, PMCEID0_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCEID0_EL0;
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCEID0_EL0;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCEID0_EL0;
  endif
elsif PSTATE.EL == EL3 then
  return PMCEID0_EL0;
end

PMCEID1_EL0, Performance Monitors Common Event Identification register 1

The PMCEID1_EL0 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the ranges $0x0020$ to $0x003F$ and $0x4020$ to $0x403F$.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

For more information about the common events and the use of the PMCEID<n>_EL0 registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID1[31:0].

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to AArch32 System register PMCEID3[31:0].

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to External register PMCEID1[31:0].

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to External register PMCEID3[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCEID1_EL0 are UNDEFINED.

**Attributes**

PMCEID1_EL0 is a 64-bit register.

**Field descriptions**

The PMCEID1_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| D31 | D30 | D29 | D28 | D27 | D26 | D25 | D24 | D23 | D22 | D21 | D20 | D19 | D18 | D17 | D16 | D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

**IDhi<n>, bit [n+32], for n = 0 to 31**

When ARMv8.1-PMU is implemented:

IDhi[n] corresponds to common event ($0x4020 + n$).

For each bit:
A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

**Accessing the PMCEID1_EL0**

Accesses to this register use the following encodings:

```
MRS <Xt>, PMCEID1_EL0
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
  AArch64.SystemAccessTrap(EL3, 0x18);
else
  return PMCEID1_EL0;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCEID1_EL0;
elseif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCEID1_EL0;
elseif PSTATE.EL == EL3 then
  return PMCEID1_EL0;
PMCNTENCLR_EL0, Performance Monitors Count Enable Clear register

The PMCNTENCLR_EL0 characteristics are:

**Purpose**

Disables the Cycle Count Register, PMCCNTR_EL0, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

**Configuration**

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCNTENCLR_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCNTENCLR_EL0 is a 64-bit register.

**Field descriptions**

The PMCNTENCLR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**C, bit [31]**

PMCCNTR_EL0 disable bit. Disables the cycle counter register. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter is enabled. When written, disables the cycle counter.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter disable bit for PMEVCNTR<n> EL0.
If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 is enabled. When written, disables PMEVCNTR&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### Accessing the PMCNTENCLR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, PMCNTENCLR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL0, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCNTENCLR_EL0;
  endif
else
  return PMCNTENCLR_EL0;
endif

MSR PMCNTENCLR_EL0, <Xt>
if PSTATE_EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  endif
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMCNTENCLR_EL0 = X[t];
  endif
else
  if PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
      SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMCNTENCLR_EL0 = X[t];
  endif
else
  if PSTATE_EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMCNTENCLR_EL0 = X[t];
    endif
  endif
else
  PMCNTENCLR_EL0 = X[t];
endif
PMCNTENSET_EL0, Performance Monitors Count Enable Set register

The PMCNTENSET_EL0 characteristics are:

Purpose

Enables the Cycle Count Register, PMCCNTR_EL0, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCNTENSET_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMCNTENSET_EL0 is a 64-bit register.

Field descriptions

The PMCNTENSET_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  | RES0 | P<n>, bit [n] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

PMCCNTR_EL0 enable bit. Enables the cycle counter register. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter is enabled. When written, enables the cycle counter.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 0 to 30

Event counter enable bit for PMEVCNTR<n>_EL0.
If \( N \) is less than 31, then bits [30\( :N \)] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, \( N \) is the value in \( \text{MDCR}_{EL2}.\text{HPMN} \). Otherwise, \( N \) is the value in \( \text{PMCR}_{EL0}.\text{N} \).

<table>
<thead>
<tr>
<th>( Pn )</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that ( \text{PMEVCNTR}<em>n</em>{EL0} ) is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that ( \text{PMEVCNTR}<em>n</em>{EL0} ) event counter is enabled. When written, enables ( \text{PMEVCNTR}<em>n</em>{EL0} ).</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally unknown value.

### Accessing the PMCNTENSET_EL0

Accesses to this register use the following encodings:

\[ \text{MRS <Xt>, PMCNTENSET_EL0} \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if \( \text{PSTATE.EL} == \text{EL0} \) then
  if \( \text{!ELUsingAArch32(EL1)} \&\& \text{PMUSERENR EL0.EN} == '0' \) then
    \( \text{AArch64.SystemAccessTrap(EL2, 0x18)} \);
  else
    \( \text{AArch64.SystemAccessTrap(EL1, 0x18)} \);
  elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL1)} \&\& \text{HCR EL2.<E2H,TGE> != '1'} \) \&\& \( \text{!HaveEL(EL3)} \\| \text{SCR EL3.FGTEn == '1'} \) \&\& \( \text{HDFGRTR EL2.PMCNTEN == '1'} \) then
    \( \text{AArch64.SystemAccessTrap(EL2, 0x18)} \);
  else
    \( \text{AArch64.SystemAccessTrap(EL3, 0x18)} \);
  end
else
  return PMCNTENSET_EL0;
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
  if \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{(!HaveEL(EL3)} \| \text{SCR EL3.FGTEn == '1'} \) \&\& \( \text{HDFGRTR EL2.PMCNTEN == '1'} \) then
    \( \text{AArch64.SystemAccessTrap(EL2, 0x18)} \);
  else
    return PMCNTENSET_EL0;
  end
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
  if \( \text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{MDCR EL3.TPM == '1'} \) then
    \( \text{AArch64.SystemAccessTrap(EL3, 0x18)} \);
  else
    return PMCNTENSET_EL0;
  end
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
  return PMCNTENSET_EL0;

\[ \text{MSR PMCNTENSET_EL0, <Xt>} \]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t];
    end if
    PMCNTENSET_EL0 = X[t];
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t];
    end if
else
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENSET_EL0 = X[t];
        end if
    elsif PSTATE.EL == EL3 then
        PMCNTENSET_EL0 = X[t];
    else
        PMCNTENSET_EL0 = X[t];
    end if
else
    if PSTATE.EL == EL0 then
        if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            else
                AArch64.SystemAccessTrap(EL1, 0x18);
            end if
        elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENSET_EL0 = X[t];
        end if
        PMCNTENSET_EL0 = X[t];
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMCNTENSET_EL0 = X[t];
    end if
    PMCNTENSET_EL0 = X[t];
else
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENSET_EL0 = X[t];
        end if
    elseif PSTATE.EL == EL3 then
        PMCNTENSET_EL0 = X[t];
    else
        PMCNTENSET_EL0 = X[t];
    end if
end if

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

**Purpose**

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

**Configuration**

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCR[31:0].

AArch64 System register PMCR_EL0 bits [7:0] are architecturally mapped to External register PMCR_EL0[7:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCR_EL0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCR_EL0 is a 64-bit register.

**Field descriptions**

The PMCR_EL0 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     | IMP | IDCODE | N | RES0 |     | LP | LC | DP | X | D | C | P | E |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:32]**

Reserved, RES0.

**IMP, bits [31:24]**

Implementer code. This field has an IMPLEMENTATION DEFINED value.

If this field is zero, then PMCR_EL0.IDCODE is RES0 and software must use the MIDR_EL1 to identify the PE.

Otherwise, this field and PMCR_EL0.IDCODE identifies the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as MIDR_EL1.Implementer.

Access to this field is RO.

**IDCODE, bits [23:16]**

When PMCR_EL0.IMP != 0x00:

Identification code. This field has an IMPLEMENTATION DEFINED value.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

Access to this field is RO.
Otherwise:

Reserved, RES0.

\textbf{N, bits [15:11]}

Indicates the number of event counters implemented. This value is in the range of 0b00000-0b11111. If the value is 0b00000 then only \texttt{PMCCNTR_EL0} is implemented. If the value is 0b11111 \texttt{PMCCNTR_EL0} and 31 event counters are implemented.

When EL2 is implemented and enabled for the current Security state, reads of this field from EL1 and EL0 return the value of \texttt{MDCR_EL2.HPMN}.

Access to this field is \textbf{RO}.

\textbf{Bits [10:8]}

Reserved, RES0.

\textbf{LP, bit [7]}

When ARMv8.5-PMU is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

<table>
<thead>
<tr>
<th>LP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR(&lt;n&gt;)_EL0{31:0}.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counter overflow on increment that causes unsigned</td>
</tr>
<tr>
<td></td>
<td>overflow of PMEVCNTR(&lt;n&gt;)_EL0{63:0}.</td>
</tr>
</tbody>
</table>

If EL2 is implemented and \texttt{MDCR_EL2.HPMN} or \texttt{HDCR.HPMN} is less than \texttt{PMCR_EL0.N}, this bit does not affect the operation of event counters in the range [HDCR.HPMN..(PMCR_EL0.N-1)] or [MDCR_EL2.HPMN..(PMCR_EL0.N-1)].

\textbf{Note}

The effect of \texttt{MDCR_EL2.HPMN} or \texttt{HDCR.HPMN} on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, and regardless of whether EL2 is enabled in the current Security state. For more information, see the description of \texttt{MDCR_EL2.HPMN} or \texttt{HDCR.HPMN}.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

\textbf{LC, bit [6]}

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

<table>
<thead>
<tr>
<th>LC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0{31:0}.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counter overflow on increment that causes unsigned</td>
</tr>
<tr>
<td></td>
<td>overflow of PMCCNTR_EL0{63:0}.</td>
</tr>
</tbody>
</table>

Arm deprecates use of \texttt{PMCR_EL0.LC} = 0.

In an AArch64 only implementation, this field is RES1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
DP, bit [5]

When HaveEL(EL3) or (ARMv8.1-PMU is implemented and HaveEL(EL2)):

Disable cycle counter when event counting is prohibited.

<table>
<thead>
<tr>
<th>DP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting by PMCCNTR_EL0 is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>When event counting for counters in the range [0..(MDCR_EL2.HPMN-1)] is prohibited, cycle counting by PMCCNTR_EL0 is disabled.</td>
</tr>
</tbody>
</table>

For more information about the interaction between the Performance Monitors and EL3, see ‘Effect of EL3 and EL2’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes an PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

<table>
<thead>
<tr>
<th>X</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not export events.</td>
</tr>
<tr>
<td>0b1</td>
<td>Export events where not prohibited.</td>
</tr>
</tbody>
</table>

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device, for example to an OPTIONAL PE trace unit.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

When AArch32 is supported at any Exception level:

Clock divider.

<table>
<thead>
<tr>
<th>D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When enabled, PMCCNTR_EL0 counts every clock cycle.</td>
</tr>
<tr>
<td>0b1</td>
<td>When enabled, PMCCNTR_EL0 counts once every 64 clock cycles.</td>
</tr>
</tbody>
</table>

If PMCR_EL0.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
C, bit [2]

Cycle counter reset. The effects of writing to this bit are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset PMCCNTR_EL0 to zero.</td>
</tr>
</tbody>
</table>

This bit is always RAZ.

Note

Resetting PMCCNTR_EL0 does not change the cycle counter overflow bit.

The value of PMCR_EL0.LC is ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is WO.

P, bit [1]

Event counter reset. The effects of writing to this bit are:

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset all event counters accessible in the current Exception level, not including PMCCNTR_EL0, to zero.</td>
</tr>
</tbody>
</table>

This bit is always RAZ.

In EL0 and EL1:

- If EL2 is implemented and enabled in the current Security state, and MDCR_EL2.HPMN is less than PMCR_EL0.N, a write of 1 to this bit does not reset event counters in the range [MDCR_EL2.HPMN..(PMCR_EL0.N-1)].
- If EL2 is not implemented, EL2 is disabled in the current Security state, or MDCR_EL2.HPMN equals PMCR_EL0.N, a write of 1 to this bit resets all the event counters.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

Note

Resetting the event counters does not change the event counter overflow bits.

If ARMv8.5-PMU is implemented, the values of MDCR_EL2.HLP and PMCR_EL0.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is WO.

E, bit [0]

Enable.

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All event counters in the range [0..(PMN-1)] and PMCCNTR_EL0 are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>All event counters in the range [0..(PMN-1)] and PMCCNTR_EL0, are enabled by PMCNTENSET_EL0.</td>
</tr>
</tbody>
</table>

If EL2 is implemented then:

- If EL2 is using AArch32, PMN is HDCR.HPMN.
- If EL2 is using AArch64, PMN is MDCR_EL2.HPMN.
• If PMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [PMN..(PMCR_EL0.N-1)].

If EL2 is not implemented, PMN is PMCR_EL0.N.

Note

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, and regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to 0.

Accessing the PMCR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, PMCR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMCR_EL0;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMCR_EL0;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMCR_EL0;
  elsif PSTATE.EL == EL3 then
    return PMCR_EL0;

MSR PMCR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMCR_EL0 = X[t];
  else if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HDFGWTR_EL2.PMCR_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMCR_EL0 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMCR_EL0 = X[t];
  elsif PSTATE.EL == EL3 then
    PMCR_EL0 = X[t];
PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

**Purpose**

Holds event counter n, which counts events, where n is 0 to 30.

**Configuration**

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVCNTR<n>[31:0].

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to External register PMEVCNTR<n>_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMEVCNTR<n>_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMEVCNTR<n>_EL0 is a 64-bit register.

**Field descriptions**

The PMEVCNTR<n>_EL0 bit assignments are:

**When ARMv8.5-PMU is implemented:**

| 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 | Event counter n |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Event counter n |

**Bits [63:0]**

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

| 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 | RES0 |
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | Event counter n |

**Bits [63:32]**

Reserved, RES0.
Bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMEVCNTR<n>_EL0

PMEVCNTR<n>_EL0 can also be accessed by using PMXEVCNTR_EL0 with PMSELR_EL0 SEL set to the value of <n>.

If ARMv8.6-FGT is implemented, and EL2 is implemented and enabled in the current Security state, for reads and writes at EL1 and permitted reads and writes at EL0:

- If <n> is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, if <n> is greater than or equal to MDCR_EL2.HPMN, the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:

If <n> is greater than or equal to the number of accessible counters, reads and writes of PMEVCNTR<n>_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.{ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible counters. Otherwise, the number of accessible counters is the number of implemented counters. See MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings:

MRS <Xt>, PMEVCNTR<n>_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b10:n[4:3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
      SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
    endif
  endif
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
    endif
  else
    if PSTATE.EL == EL2 then
      if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
      endif
    else
      if PSTATE.EL == EL3 then
        return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
      endif
    endif
  endif
endif

MSR PMEVCNTR<n>_EL0, <Xt>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
else
  if PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
else
  if PSTATE.EL == EL3 then
    PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
endif
The PMEVTYPER<n>_EL0 characteristics are:

**Purpose**

Configures event counter n, where n is 0 to 30.

**Configuration**

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVTYPER<n>[31:0].

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to External register PMEVTYPER<n>_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMEVTYPER<n>_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMEVTYPER<n>_EL0 is a 64-bit register.

**Field descriptions**

The PMEVTYPER<n>_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| P  | U  | NSK | NSK | NSH | M  | MT | SH | T  | RES0 | evtCount[15:10] | evtCount[9:0] |

**Bits [63:32]**

Reserved, RES0.

**P, bit [31]**

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMEVTYPER<n>_EL0.NSK bit.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events in EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in EL1.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**U, bit [30]**

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMEVTYPER<n>_EL0.NSU bit.
### NSK, bit [29]

**When HaveEL(EL3):**

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

- If the value of this bit is equal to the value of the PMEVTYPER<\(n\)>_EL0.P bit, events in Non-secure EL1 are counted.
- Otherwise, events in Non-secure EL1 are not counted.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### NSU, bit [28]

**When HaveEL(EL3):**

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

- If the value of this bit is equal to the value of the PMEVTYPER<\(n\)>_EL0.U bit, events in Non-secure EL0 are counted.
- Otherwise, events in Non-secure EL0 are not counted.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### NSH, bit [27]

**When HaveEL(EL2):**

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

- If Secure EL2 is implemented, counting in Secure EL2 is further controlled by the PMEVTYPER<\(n\)>_EL0.SH bit.

<table>
<thead>
<tr>
<th>NSH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not count events in EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count events in EL2.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### M, bit [26]

**When HaveEL(EL3):**

Secure EL3 filtering bit.

- If the value of this bit is equal to the value of the PMEVTYPER<\(n\)>_EL0.P bit, events in Secure EL3 are counted.
Otherwise, events in Secure EL3 are not counted.

Most applications can ignore this field and set its value to 0b0.

Note
This field is not visible in the AArch32 PMEVTYPER<n> System register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

MT, bit [25]
When ARMv8.6-MTPMU is implemented or IMPLEMENTATION DEFINED multi-threaded PMU Extension is implemented:

Multithreading.

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events only on controlling PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count events from any PE with the same affinity at level 1 and above as this PE.</td>
</tr>
</tbody>
</table>

Note
- When the lowest level of affinity consists of logical PEs that are implemented using a multi-threading type approach, an implementation is described as multi-threaded. That is, the performance of PEs at the lowest affinity level is highly interdependent.
- Events from a different thread of a multithreaded implementation are not attributable to the thread counting the event.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

SH, bit [24]
When ARMv8.4-SecEL2 is implemented:

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMEVTYPER<n>_EL0.NSH bit, events in Secure EL2 are counted.

Otherwise, events in Secure EL2 are not counted.

Note
This field is not visible in the AArch32 PMEVTYPER<n> System register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

T, bit [23]
When TME is implemented:

Transactional state filtering bit. Controls counting in Transactional state. The possible values of this bit are:

<table>
<thead>
<tr>
<th>T</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit has no effect on the filtering of events.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in Transactional state.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [22:16]**

Reserved, RES0.

**evtCount[15:10], bits [15:10]**

When ARMv8.1-PMU is implemented:

Extension to evtCount[9:0]. See evtCount[9:0] for more details.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**evtCount[9:0], bits [9:0]**

Event to count. The event number of the event that is counted by event counter `PMEVCNTR<n>_EL0`.

Software must program this field with an event that is supported by the PE being programmed.

The ranges of event numbers allocated to each type of event are shown in Allocation of the PMU event number space.

If evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- If 16-bit evtCount is implemented, for the range 0x4000 to 0x403F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED events, it is UNPREDICTABLE what event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

**Note**

UNPREDICTABLE means the event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include an event from a set of common IMPLEMENTATION DEFINED events, then no event is counted and the value read back on evtCount is the value written.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMEVTYPER<n>_EL0**

PMEVTYPER<n>_EL0 can also be accessed by using `PMXEVTPTER_EL0` with `PMSELR_EL0` SEL set to n.
If ARMv8.6-FGT is implemented, and EL2 is implemented and enabled in the current Security state, for reads and writes at EL1 and permitted reads and writes at EL0:

- If <n> is an unimplemented event counter, the access is **UNDEFINED**.
- Otherwise, if <n> is greater than or equal to **MDCR_EL2.HPMN**, the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:

If <n> is greater than or equal to the number of accessible counters, reads and writes of PMEVTYPER<n> _EL0 are **CONSTRAINED UNPREDICTABLE**, and the following behaviors are permitted:

- Accesses to the register are **UNDEFINED**.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

**Note**

In EL0, an access is permitted if it is enabled by **PMUSERENR_EL0.EN**.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, **MDCR_EL2.HPMN** identifies the number of accessible counters. Otherwise, the number of accessible counters is the number of implemented counters. See **MDCR_EL2.HPMN** for more details.

Accesses to this register use the following encodings:

**MRS <Xt>, PMEVTYPER<n> _EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b11:n[4:3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    else
        if !ELUsingAArch32(EL1) && SCR_EL3.FGTEn == '1' && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        endif
    endelse
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    endif
else
    return PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
endelse
```

```c
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
    endif
else
    return PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
endelse
```

```c
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
    endif
else
    return PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
endelse
```

```c
elsif PSTATE.EL == EL3 then
    return PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
endelse
```
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMEVTPYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
endif

if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMEVTPYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
endif

if PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMEVTPYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
  endif
endif

if PSTATE.EL == EL3 then
  PMEVTPYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
endif
PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR_EL1 characteristics are:

**Purpose**

Disables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR_EL0, and the event counters PMEVCNTR<n>_EL0. Reading the register shows which overflow interrupt requests are enabled.

**Configuration**

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMINTENCLR_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMINTENCLR_EL1 is a 64-bit register.

**Field descriptions**

The PMINTENCLR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>C, bit [31]</td>
</tr>
<tr>
<td>61</td>
<td>PMCCNTR_EL0 overflow interrupt request disable bit. Possible values are:</td>
</tr>
<tr>
<td>60</td>
<td>0b0</td>
</tr>
<tr>
<td>59</td>
<td>0b1</td>
</tr>
<tr>
<td>58</td>
<td>P&lt;n&gt;, bit [n], for n = 0 to 30</td>
</tr>
</tbody>
</table>

**C, bit [31]**

PMCCNTR_EL0 overflow interrupt request disable bit. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request.</td>
</tr>
</tbody>
</table>

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow interrupt request disable bit for PMEVCNTR<n>_EL0.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.
<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that the PMEVCNTR&lt;n&gt;_EL0 event counter interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that the PMEVCNTR&lt;n&gt;_EL0 event counter interrupt request is enabled. When written, disables the PMEVCNTR&lt;n&gt;_EL0 interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMINTENCLR_EL1**

Accesses to this register use the following encodings:

**MRS <Xt>, PMINTENCLR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMINTEN == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMINTENCLR_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMINTENCLR_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    PMINTENCLR_EL1 = X[t];

**MSR PMINTENCLR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMINTEN == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMINTENCLR_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMINTENCLR_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    PMINTENCLR_EL1 = X[t];
PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear register
PMINTENSET_EL1, Performance Monitors Interrupt Enable Set register

The PMINTENSET_EL1 characteristics are:

**Purpose**

Enables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR_EL0, and the event counters PMEVCNTR<n>_EL0. Reading the register shows which overflow interrupt requests are enabled.

**Configuration**

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENSET[31:0].

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMINTENSET_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMINTENSET_EL1 is a 64-bit register.

**Field descriptions**

The PMINTENSET_EL1 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

C, bit [31]

PMCCNTR_EL0 overflow interrupt request enable bit. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow interrupt request enable bit for PMEVCNTR<n>_EL0.
If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that the PMEVCNTR&lt;n&gt;_EL0 event counter interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that the PMEVCNTR&lt;n&gt;_EL0 event counter interrupt request is enabled. When written, enables the PMEVCNTR&lt;n&gt;_EL0 interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### Accessing the PMINTENSET_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMINTENSET_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENSET_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMINTENSET_EL1;
    elsif PSTATE.EL == EL3 then
        return PMINTENSET_EL1;
```

```
MSR PMINTENSET_EL1, <Xt>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENSET_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMINTENSET_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMINTENSET_EL1 = X[t];
PMMIR_EL1, Performance Monitors Machine Identification Register

The PMMIR_EL1 characteristics are:

**Purpose**

Describes Performance Monitors parameters specific to the implementation to software.

**Configuration**

This register is present only when ARMv8.4-PMU is implemented. Otherwise, direct accesses to PMMIR_EL1 are UNDEFINED.

**Attributes**

PMMIR_EL1 is a 64-bit register.

**Field descriptions**

The PMMIR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:8]**

Reserved, RES0.

**SLOTS, bits [7:0]**

Operation width. The largest value by which the STALL_SLOT event might increment by in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

**Accessing the PMMIR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, PMMIR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.PMMIR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMMIR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMMIR_EL1;
elsif PSTATE.EL == EL3 then
  return PMMIR_EL1;
The PMOVSCLR_EL0 characteristics are:

**Purpose**

Contains the state of the overflow bit for the Cycle Count Register, `PMCCNTR_EL0`, and each of the implemented event counters `PMEVCNTR<n>`. Writing to this register clears these bits.

**Configuration**

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register `PMOVSR[31:0]`.

AArch64 System register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to External register `PMOVSLR_EL0[31:0]`.

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMOVSCLR_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMOVSCLR_EL0 is a 64-bit register.

**Field descriptions**

The PMOVSCLR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>C</td>
</tr>
<tr>
<td>61</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**C, bit [31]**

Cycle counter overflow clear bit.

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter has overflowed since this bit was last cleared. When written, clears the cycle counter overflow bit to 0.</td>
</tr>
</tbody>
</table>

`PMCR_EL0.LC` controls whether an overflow is detected from unsigned overflow of `PMCCNTR_EL0[31:0]` or unsigned overflow of `PMCCNTR_EL0[63:0]`.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
P<n>, bit [n], for n = 0 to 30

Event counter overflow clear bit for PMEVCNTR<n>_EL0.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 has overflowed since this bit was last cleared. When written, clears the PMEVCNTR&lt;n&gt;_EL0 overflow bit to 0.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<n>_EL0[31:0] or unsigned overflow of PMEVCNTR<n>_EL0[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing the PMOVSLER_EL0

Accesses to this register use the following encodings:

MRS <Xt>, PMOVSLER_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOV5 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMOVSLER_EL0;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOV5 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMOVSLER_EL0;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return PMOVSLER_EL0;
  elsif PSTATE.EL == EL3 then
    return PMOVSLER_EL0;
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMOVSCLR_EL0 = X[t];
  else if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMOVSCLR_EL0 = X[t];
    else if PSTATE.EL == EL2 then
      if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        PMOVSCLR_EL0 = X[t];
      else if PSTATE.EL == EL3 then
        PMOVSCLR_EL0 = X[t];
The PMOVSSET_EL0 characteristics are:

**Purpose**

Sets the state of the overflow bit for the Cycle Count Register, PMCCNTR_EL0, and each of the implemented event counters PMEVCNTR<n>.

**Configuration**

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMOVSSET_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMOVSSET_EL0 is a 64-bit register.

**Field descriptions**

The PMOVSSET_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>P&lt;n&gt;, bit [n]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**C, bit [31]**

Cycle counter overflow set bit.

<table>
<thead>
<tr>
<th>0b0</th>
<th>0b1</th>
</tr>
</thead>
<tbody>
<tr>
<td>When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect.</td>
<td>When read, means the cycle counter has overflowed since this bit was last cleared. When written, sets the cycle counter overflow bit to 1.</td>
</tr>
</tbody>
</table>

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.
**P<n>, bit [n], for n = 0 to 30**

Event counter overflow set bit for **PMEVCNTR<n>_EL0**.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in **MDCR_EL2**. HPMN. Otherwise, N is the value in **PMCR_EL0**.N.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that <strong>PMEVCNTR&lt;n&gt;_EL0</strong> has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that <strong>PMEVCNTR&lt;n&gt;_EL0</strong> has overflowed since this bit was last cleared. When written, sets the <strong>PMEVCNTR&lt;n&gt;_EL0</strong> overflow bit to 1.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, **MDCR_EL2**.HLp and **PMCR_EL0**.LP control whether an overflow is detected from unsigned overflow of **PMEVCNTR<n>_EL0[31:0]** or unsigned overflow of **PMEVCNTR<n>_EL0[63:0]**.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMOVSSET_EL0**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b011</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    else
        AArch64.SystemAccessTrap(EL1, 0x18);
    endif
    if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL2, 0x18);
    endif
else
    AArch64.SystemAccessTrap(EL3, 0x18);
endif
```

```plaintext
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMOVSSET_EL0;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMOVSSET_EL0;
    endif
elsif PSTATE.EL == EL3 then
    return PMOVSSET_EL0;
```

PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set register
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t];
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        endif
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccess Trap(EL3, 0x18);
    else
        PMOVSSET_EL0 = X[t];
    endif
else
    if PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSSET_EL0 = X[t];
        endif
    else
        PMOVSSET_EL0 = X[t];
    endif
else
    PMOVSSET_EL0 = X[t];
endif
PMSCR_EL1, Statistical Profiling Control Register (EL1)

The PMSCR_EL1 characteristics are:

**Purpose**

Provides EL1 controls for Statistical Profiling

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSCR_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMSCR_EL1 is a 64-bit register.

**Field descriptions**

The PMSCR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>PCT</td>
</tr>
<tr>
<td>60</td>
<td>TS</td>
</tr>
<tr>
<td>59</td>
<td>PA</td>
</tr>
<tr>
<td>58</td>
<td>CX</td>
</tr>
<tr>
<td>57</td>
<td>E1SPE</td>
</tr>
<tr>
<td>56</td>
<td>E0SPE</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:8]**

Reserved, RES0.

**PCT, bits [7:6]**

When HaveEL(EL2):

Physical Timestamp. If timestamp sampling is enabled and the Profiling Buffer is owned by EL1, requests which timestamp counter value is collected.

If ARMv8.6-ECV is implemented, this is a two bit field as shown. Otherwise, bit[7] is RES0.

<table>
<thead>
<tr>
<th>PCT</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Virtual counter; CNTVCT_EL0 is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collected.</td>
<td></td>
</tr>
<tr>
<td>0b01</td>
<td>Physical counter; CNTPCT_EL0 is</td>
<td></td>
</tr>
<tr>
<td></td>
<td>collected.</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>Physical counter, CNTPCT_EL0,</td>
<td>When ARMv8.6-ECV is</td>
</tr>
<tr>
<td></td>
<td>minus CNTPCT_EL2 is collected.</td>
<td>implemented</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the Profiling Buffer owning Exception level is EL2, this field is ignored.

If the Profiling Buffer owning Exception level is EL1, this field is combined with PMSCR_EL2.PCT to determine which timestamp counter value is collected. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:

- **Cnthctl_EL2.ECV** is 0.
• **SCR_EL3**.ECVEn is 0.
• **HCR_EL2**.E2H, TGE) is (1, 1).

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Physical Timestamp. Reserved. This field reads as 0b01 and ignores writes. Software should treat this field as UNK/ SBZP.

When EL2 is not implemented, the Effective values of **CNTVOFF_EL2** and **CNTPOFF_EL2** are zero, meaning the virtual counter and physical counter have the same value.

**TS, bit [5]**

Timestamp enable.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timestamp sampling disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timestamp sampling enabled.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE if EL2 is implemented and the Profiling Buffer is owned by EL2. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**PA, bit [4]**

Physical Address sample enable.

<table>
<thead>
<tr>
<th>PA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Physical addresses are not collected.</td>
</tr>
<tr>
<td>0b1</td>
<td>Physical addresses are collected.</td>
</tr>
</tbody>
</table>

If EL2 is implemented:

• If the Profiling Buffer is owned by EL1, this bit is combined with **PMSCR_EL2**.PA to determine which address is collected. For more information, see 'Controlling the data that is collected' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
• If the Profiling Buffer is owned by EL2, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**CX, bit [3]**

**CONTEXTIDR_EL1** sample enable.

<table>
<thead>
<tr>
<th>CX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>CONTEXTIDR_EL1</strong> is not collected.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>CONTEXTIDR_EL1</strong> is collected.</td>
</tr>
</tbody>
</table>

If EL2 is implemented and enabled in the current Security state when an operation is sampled:

• If the PE is at EL2, this bit is ignored by the PE.
• If **HCR_EL2**.TGE == 1, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bit [2]**

Reserved, RES0.
E1SPE, bit [1]

EL1 Statistical Profiling Enable.

<table>
<thead>
<tr>
<th>E1SPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sampling disabled at EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sampling enabled at EL1.</td>
</tr>
</tbody>
</table>

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when $\text{HCR\_EL2.TGE == \text{1}}$.

On a Warm reset, this field resets to an architecturally \text{UNKNOWN} value.

E0SPE, bit [0]

EL0 Statistical Profiling Enable. Controls sampling at EL0 when $\text{HCR\_EL2.TGE == \text{0}}$ or if EL2 is disabled or not implemented.

<table>
<thead>
<tr>
<th>E0SPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sampling disabled at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sampling enabled at EL0.</td>
</tr>
</tbody>
</table>

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when $\text{HCR\_EL2.TGE == \text{1}}$.

On a Warm reset, this field resets to an architecturally \text{UNKNOWN} value.

Accessing the PMSCR\_EL1

Accesses to this register use the following encodings:

```
MRS <Xt>, PMSCR\_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR\_EL3.\text{FGTE}\_n == \text{1}) && HDFGRTR\_EL2.PMSCR\_EL1 == \text{'1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & MDCR\_EL2.TPMS == \text{'1'} then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & SCR\_EL3.\text{NS} == \text{'0'} & MDCR\_EL3.\text{NSPB} != \text{'01'} then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & SCR\_EL3.\text{NS} == \text{'1'} & MDCR\_EL3.\text{NSPB} != \text{'11'} then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & HCR\_EL2.<\text{NV2,NV1,NV}> == \text{'111'} then
    return NVMem[0x828];
  else
    return PMSCR\_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) & !ELUsingAArch32(EL3) & SCR\_EL3.\text{NS} == \text{'0'} & MDCR\_EL3.\text{NSPB} != \text{'01'} then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & SCR\_EL3.\text{NS} == \text{'1'} & MDCR\_EL3.\text{NSPB} != \text{'11'} then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR\_EL2.\text{E2H} == \text{'1'} then
    return PMSCR\_EL2;
  else
    return PMSCR\_EL1;
elsif PSTATE.EL == EL3 then
  return PMSCR\_EL1;
PMSCR_EL1, Statistical Profiling Control Register (EL1)

MSR PMSCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSCR_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
      NVMem[0x828] = X[t];
   else
      PMSCR_EL1 = X[t];
   elsif PSTATE.EL == EL2 then
      if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
         return NVMem[0x828];
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
         return PMSCR_EL1;
      else
         UNDEFINED;
   elsif PSTATE.EL == EL3 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
         if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
         elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
         else
            return PMSCR_EL1;
         else
            UNDEFINED;
      elsif PSTATE.EL == EL4 then
         if EL2Enabled() && HCR_EL2.E2H == '1' then
            return PMSCR_EL1;
         else
            UNDEFINED;
      else
         UNDEFINED;
   else
      UNDEFINED;

MRS <Xt>, PMSCR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
      return NVMem[0x828];
   elsif EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   if EL2Enabled() && HCR_EL2.E2H == '1' then
      if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         return PMSCR_EL1;
      else
         UNDEFINED;
elsif PSTATE.EL == EL3 then
   if EL2Enabled() && HCR_EL2.E2H == '1' then
      return PMSCR_EL1;
   else
      UNDEFINED;
else
   UNDEFINED;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x828] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMSCR_EL1 = X[t];
    else
      UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    PMSCR_EL1 = X[t];
  else
    UNDEFINED;

PMSCR_EL2, Statistical Profiling Control Register (EL2)

The PMSCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Statistical Profiling

Configuration

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSCR_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMSCR_EL2 is a 64-bit register.

Field descriptions

The PMSCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits [63:8]</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>PCT, bits [7:6]</td>
<td>Physical Timestamp. If timestamp sampling is enabled, determines which counter is collected. The behavior depends on the Profiling Buffer owning Exception level.</td>
</tr>
</tbody>
</table>

If ARMv8.6-ECV is implemented, this is a two bit field as shown. Otherwise, bit[7] is RES0.

<table>
<thead>
<tr>
<th>PCT</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Virtual counter, CNTVCT_EL0, is collected.</td>
<td></td>
</tr>
<tr>
<td>0b01</td>
<td>If the Profiling Buffer owning Exception level is EL2: Physical counter, CNTPCT_EL0, is collected. If the Profiling Buffer owning Exception level is EL1: Timestamp value is selected by PMSCR_EL1.PCT.</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>If the Profiling Buffer owning Exception level is EL2: Physical counter, CNTPCT_EL0, minus CNTPOFF_EL2 is collected. If the Profiling Buffer owning Exception level is EL1: If PMSCR_EL1.PCT == 0b00, the virtual counter, CNTVCT_EL0, is collected. Otherwise, the physical counter, CNTPCT_EL0, minus CNTPOFF_EL2 is collected.</td>
<td>When ARMv8.6-ECV is implemented</td>
</tr>
</tbody>
</table>
When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:

- **CNTHCTL_EL2**.ECV is 0.
- **SCR_EL3**.ECVEn is 0.
- **HCR_EL2**.({E2H, TGE}) is {1, 1}.

If EL2 is not implemented or EL2 is disabled in the current Security state, the PE behaves as if this field is set to 0b01, other than for a direct read of the register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**TS, bit [5]**

Timestamp Enable.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timestamp sampling disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timestamp sampling enabled.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when any of the following are true:

- The Profiling Buffer owning Exception level is EL1.
- In Secure state, and either ARMv8.4-SecEL2 is not implemented or Secure EL2 is disabled.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**PA, bit [4]**

Physical Address Sample Enable.

<table>
<thead>
<tr>
<th>PA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Physical addresses are not collected.</td>
</tr>
<tr>
<td>0b1</td>
<td>Physical addresses are collected.</td>
</tr>
</tbody>
</table>

If the Profiling Buffer owning Exception level is EL1, and EL2 is enabled in the current Security state, this bit is combined with **PMSCR_EL1**.PA to determine which address is collected.

If EL2 is not implemented or EL2 is disabled in the current Security state, the PE ignores the value of this bit and behaves as if this bit is set to 1, other than for a direct read of the register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**CX, bit [3]**

**CONTEXTIDR_EL2** Sample Enable.

<table>
<thead>
<tr>
<th>CX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>CONTEXTIDR_EL2</strong> is not collected.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>CONTEXTIDR_EL2</strong> is collected.</td>
</tr>
</tbody>
</table>

If EL2 is not implemented or EL2 is disabled in the current Security state, the PE ignores the value of this bit.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bit [2]**

Reserved, RES0.

**E2SPE, bit [1]**

EL2 Statistical Profiling Enable.
E2SPE, Meaning

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sampling disabled at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sampling enabled at EL2.</td>
</tr>
</tbody>
</table>

This bit is res0 if MDCR_EL2.E2PB != 0b00.

If EL2 is disabled in the current Security state, this bit is ignored by the PE.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

E0HSPE, bit [0]

EL0 Statistical Profiling Enable.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sampling disabled at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sampling enabled at EL0.</td>
</tr>
</tbody>
</table>

If MDCR_EL2.E2PB != 0b00, this bit is res0.

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when HCR_EL2.TGE == 0.

On a Warm reset, this field resets to an architectually UNKNOWN value.

Accessing the PMSCR_EL2

Accesses to this register use the following encodings:

MRS <Xt>, PMSCR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMSCR_EL2;
elsif PSTATE.EL == EL3 then
  return PMSCR_EL2;

MSR PMSCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSCR_EL2 = X[t];
        end
    elsif PSTATE.EL == EL3 then
        PMSCR_EL2 = X[t];
    end
MRS <Xt>, PMSCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x828];
    else
        return PMSCR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return PMSCR_EL2;
    else
        return PMSCR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return PMSCR_EL1;
MSR PMSCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.PMSCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        PMSCR_EL2 = X[t];
    else
        PMSCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        PMSCR_EL2 = X[t];
    else
        PMSCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSCR_EL1 = X[t];
PMSELR_EL0, Performance Monitors Event Counter Selection Register

The PMSELR_EL0 characteristics are:

**Purpose**

Selects the current event counter `PMEVCNTR<n>_EL0` or the cycle counter, CCNT.

PMSELR_EL0 is used in conjunction with `PMXEVTYPER_EL0` to determine the event that increments a selected event counter, and the modes and states in which the selected counter increments.

It is also used in conjunction with `PMXEV CNTR_EL0` to determine the value of a selected event counter.

**Configuration**

AArch64 System register PMSELR_EL0 bits [31:0] are architecturally mapped to AArch32 System register `PMSEL[31:0]`.

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMSELR_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMSELR_EL0 is a 64-bit register.

**Field descriptions**

The PMSELR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

RES0

RES0

SEL

**Bits [63:5]**

Reserved, RES0.

**SEL, bits [4:0]**

Selects event counter, `PMEVCNTR<n>_EL0`, where n is the value held in this field. This value identifies which event counter is accessed when a subsequent access to `PMXEVTYPER_EL0` or `PMXEV CNTR_EL0` occurs.

This field can take any value from 0 (0b00000) to (PMCR.N)-1, or 31 (0b11111).

When PMSELR_EL0.SEL is 0b11111, it selects the cycle counter and:

- A read of the `PMXEVTYPER_EL0` returns the value of `PMCCFILTR_EL0`.
- A write of the `PMXEVTYPER_EL0` writes to `PMCCFILTR_EL0`.
- A read or write of `PMXEV CNTR_EL0` has CONSTRAINED UNPREDICTABLE effects. See `PMXEV CNTR_EL0` for more details.

For details of the results of accesses to the event counters, see `PMXEVTYPER_EL0` and `PMXEV CNTR_EL0`.

For information about the number of counters accessible at each Exception level, see `MDCR_EL2_HPMN`.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
Accessing the PMSELR_EL0

Accesses to this register use the following encodings:

MRS <Xt>, PMSELR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSELR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMSELR_EL0;
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSELR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMSELR_EL0;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMSELR_EL0;
  endif
elsif PSTATE.EL == EL3 then
  return PMSELR_EL0;
elsif PSTATE.EL == EL4 then
  return PMSELR_EL0;

MSR PMSELR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b111</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMSELR_EL0 = X[t];
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMSELR_EL0 = X[t];
  else
    if PSTATE.EL == EL1 then
      if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        PMSELR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
      if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        PMSELR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
      PMSELR_EL0 = X[t];
PMSEVFR_EL1, Sampling Event Filter Register

The PMSEVFR_EL1 characteristics are:

**Purpose**

Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if E[3] and E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 set (TLB walk) are recorded.

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSEVFR_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMSEVFR_EL1 is a 64-bit register.

**Field descriptions**

The PMSEVFR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Event $&lt;z&gt;$ is ignored.</td>
</tr>
<tr>
<td>62</td>
<td>Do not record samples that have event $&lt;z&gt; == 0.$</td>
</tr>
</tbody>
</table>

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE == 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bits [47:32]**

Reserved, RAZ/WI.

**E[$<y>$], bit [y], for y = 24 to 31**

E[$<y>$] is the event filter for event $<y>$. If event $<y>$ is not implemented, or filtering on event $<y>$ is not supported, the corresponding bit is RAZ/WI.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Event $&lt;y&gt;$ is ignored.</td>
</tr>
<tr>
<td>30</td>
<td>Do not record samples that have event $&lt;y&gt; == 0.$</td>
</tr>
</tbody>
</table>

This field is ignored by the PE when PMSFCR_EL1.FE == 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.
An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSFCR_EL1.FE == 0

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:19]

Reserved, RAZ/WI.

E[18], bit [18]

When ARMv8.3-SPE is implemented and SVE is implemented:

Empty predicate.

<table>
<thead>
<tr>
<th>E[18]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Empty predicate event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Empty predicate event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]

When ARMv8.3-SPE is implemented and SVE is implemented:

Partial predicate.

<table>
<thead>
<tr>
<th>E[17]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Partial predicate event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Partial predicate event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[16], bit [16]

When TME is implemented:

Transactional

<table>
<thead>
<tr>
<th>E[16]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Transactional event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Transactional event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RAZ/WI.

**E[<x>], bit [x], for x = 12 to 15**

E[<x>] is the event filter for event <x>. If event <x> is not implemented, or filtering on event <x> is not supported, the corresponding bit is RAZ/WI.

<table>
<thead>
<tr>
<th>E[&lt;x&gt;]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event &lt;x&gt; is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have event &lt;x&gt; == 0.</td>
</tr>
</tbody>
</table>

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when PMSEVFR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**E[11], bit [11]**

**When ARMv8.3-SPE is implemented:**

Alignment.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Alignment event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Alignment event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSEVFR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RAZ/WI.

**Bits [10:8]**

Reserved, RAZ/WI.

**E[7], bit [7]**

Mispredicted.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Mispredicted event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Mispredicted event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSEVFR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bit [6]**

Reserved, RAZ/WI.

**E[5], bit [5]**

TLB walk.
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TLB walk event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the TLB walk event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bit [4]**

Reserved, RAZ/WI.

**E[3], bit [3]**

Level 1 data or unified cache refill.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Level 1 data or unified cache refill event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Level 1 data or unified cache refill event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bit [2]**

Reserved, RAZ/WI.

**E[1], bit [1]**

*When the PE supports sampling of speculative instructions:*

Architecturally retired.

When the PE supports sampling of speculative instructions:

<table>
<thead>
<tr>
<th>E[1]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Architecturally retired event is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not record samples that have the Architecturally retired event == 0.</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FE == 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

*Otherwise:*

Reserved, UNKNOWN.

**Bit [0]**

Reserved, RAZ/WI.

**Accessing the PMSEVFR_EL1**

Accesses to this register use the following encodings:
PMSEVFR_EL1, Sampling Event Filter Register

MRS <Xt>, PMSEVFR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.PMSEVFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
        MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
        MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x830];
    else
        return PMSEVFR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
            MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
            MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSEVFR_EL1;
        elsif PSTATE.EL == EL3 then
            PMSEVFR_EL1 = X[t];

MSR PMSEVFR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.PMSEVFR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
        MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
        MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x830];
    else
        PMSEVFR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
            MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
            MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSEVFR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSEVFR_EL1 = X[t];
PMSFCR_EL1, Sampling Filter Control Register

The PMSFCR_EL1 characteristics are:

**Purpose**

Controls sample filtering. The filter is the logical AND of the FL, FT and FE bits. For example, if FE == 1 and FT == 1 only samples including the selected operation types and the selected events will be recorded.

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSFCR_EL1 are undefined. RW fields in this register reset to architecturally unknown values.

**Attributes**

PMSFCR_EL1 is a 64-bit register.

**Field descriptions**

The PMSFCR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>ST</td>
<td>Store filter enable</td>
</tr>
<tr>
<td>17</td>
<td>LD</td>
<td>Load filter enable</td>
</tr>
</tbody>
</table>

**Bits [63:19]**

Reserved, RES0.

**ST, bit [18]**

Store filter enable

<table>
<thead>
<tr>
<th>ST</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not record store operations</td>
</tr>
<tr>
<td>0b1</td>
<td>Record all store operations, including vector stores and all atomic operations</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

On a Warm reset, this field resets to an architecturally unknown value.

**LD, bit [17]**

Load filter enable

<table>
<thead>
<tr>
<th>LD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not record load operations</td>
</tr>
<tr>
<td>0b1</td>
<td>Record all load operations, including vector loads and atomic operations that return data</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

On a Warm reset, this field resets to an architecturally unknown value.
**B, bit [16]**

Branch filter enable

<table>
<thead>
<tr>
<th>B</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not record branch and exception return operations</td>
</tr>
<tr>
<td>0b1</td>
<td>Record all branch and exception return operations</td>
</tr>
</tbody>
</table>

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [15:3]**

Reserved, **RES0**.

**FL, bit [2]**

Filter by latency

<table>
<thead>
<tr>
<th>FL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Latency filtering disabled</td>
</tr>
<tr>
<td>0b1</td>
<td>Latency filtering enabled. Samples with a total latency less than PMSLATFR_EL1.MINLAT will not be recorded</td>
</tr>
</tbody>
</table>

If this field is set to 1 and PMSLATFR_EL1.MINLAT is set to zero, it is **CONSTRAINED UNPREDICTABLE** whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FL is set to 0.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**FT, bit [1]**

Filter by operation type. The filter is the logical OR of the ST, LD and B bits. For example, if LD and ST are both set, both load and store operations are recorded

<table>
<thead>
<tr>
<th>FT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Type filtering disabled</td>
</tr>
<tr>
<td>0b1</td>
<td>Type filtering enabled. Samples not one of the selected operation types will not be recorded</td>
</tr>
</tbody>
</table>

If this field is set to 1 and the PMSFCR_EL1.{ST, LD, B} bits are all set to zero, it is **CONSTRAINED UNPREDICTABLE** whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is set to 0.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**FE, bit [0]**

Filter by event

<table>
<thead>
<tr>
<th>FE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event filtering disabled</td>
</tr>
<tr>
<td>0b1</td>
<td>Event filtering enabled. Samples not including the events selected by PMSEVFR_EL1 will not be recorded</td>
</tr>
</tbody>
</table>

If this field is set to 1 and PMSEVFR_EL1 is set to zero, it is **CONSTRAINED UNPREDICTABLE** whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FE is set to 0.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMSFCR_EL1**

Accesses to this register use the following encodings:
MRS \langle X_t \rangle, \text{PMSFCR_EL1}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSFCR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSFCR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMSFCR_EL1;

MSR PMSFCR_EL1, \langle X_t \rangle

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSFCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSFCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSFCR_EL1 = X[t];
PMSICR_EL1, Sampling Interval Counter Register

The PMSICR_EL1 characteristics are:

**Purpose**

Software must write zero to PMSICR_EL1 before enabling sample profiling for a sampling session. Software must then treat PMSICR_EL1 as an opaque, 64-bit, read/write register used for context switches only.

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSICR_EL1 are undefined. RW fields in this register reset to architecturally unknown values.

**Attributes**

The value of PMSICR_EL1 does not change whilst profiling is disabled.

**Field descriptions**

The PMSICR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECOUNT</td>
<td>[63:56]</td>
</tr>
<tr>
<td>COUNT</td>
<td>[31:0]</td>
</tr>
</tbody>
</table>

**ECOUNT, bits [63:56]**

When \( \text{PMSIDR_EL1.ERand} = 0b1 \):

Secondary sample interval counter.

When \( \text{PMSIDR_EL1.ERand} \) is 1, this field provides the secondary counter used after the primary counter reaches zero. Whilst the secondary counter is nonzero and profiling is enabled, the secondary counter decrements by 1 for each member of the sample population. The primary counter also continues to decrement since it is also nonzero. When the secondary counter reaches zero, a member of the sampling population is selected for sampling.

When \( \text{PMSIDR_EL1.ERand} \) is 0, this field is RES1.

On a Warm reset, this field resets to an architecturally unknown value.

**Otherwise:**

Reserved, RES0.

**Bits [55:32]**

Reserved, RES0.

**COUNT, bits [31:0]**

Primary sample interval counter

Provides the primary counter used for sampling.
The primary counter is reloaded when the value of this register is zero and the PE moves from a state or Exception level where profiling is disabled to a state or Exception level where profiling is enabled.

Whilst the primary counter is nonzero and sampling is enabled, the primary counter decrements by 1 for each member of the sample population.

When the counter reaches zero, the behavior depends on the values of PMSIDR_EL1.ERnd and PMSIRR_EL1.RND:

- If $\text{PMSIRR\_EL1.RND} == 0$ or $\text{PMSIDR\_EL1.ERnd} == 0$:
  - A member of the sampling population is selected for sampling
  - The primary counter is reloaded

- If $\text{PMSIRR\_EL1.RND} == 1$ and $\text{PMSIDR\_EL1.ERnd} == 1$:
  - The secondary counter is set to a random or pseudorandom value in the range $0x00$ to $0xFF$
  - The primary counter is reloaded

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

### Accessing the PMSICR_EL1

Accesses to this register use the following encodings:

**MRS <Xt>, PMSICR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsf PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSICR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    return NWMem[0x838];
else
    return PMSICR_EL1;
elsf PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return PMSICR_EL1;
elsf PSTATE.EL == EL3 then
    return PMSICR_EL1;
else
    return PMSICR_EL1;
```

**MSR PMSICR_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.PMSICR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x838] = X[t];
        else
        PMSICR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
        else
        PMSICR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSICR_EL1 = X[t];
The PMSIDR_EL1 characteristics are:

**Purpose**

Describes the Statistical Profiling implementation to software

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSIDR_EL1 are UNDEFINED.

**Attributes**

PMSIDR_EL1 is a 64-bit register.

**Field descriptions**

The PMSIDR_EL1 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | RES0 | CountSize | MaxSize | Interval | RES0 | ERnd | LDS | ArchInst | FL | FT | FE |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:20]**

Reserved, RES0.

**CountSize, bits [19:16]**

Defines the size of the counters

<table>
<thead>
<tr>
<th>CountSize</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0010</td>
<td>12-bit saturating counters</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

**MaxSize, bits [15:12]**

Defines the largest size for a single record, rounded up to a power-of-two. If this is the same as the minimum alignment (PMBIDR_EL1.Align), then each record is exactly this size

<table>
<thead>
<tr>
<th>MaxSize</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0100</td>
<td>16 bytes</td>
</tr>
<tr>
<td>0b0101</td>
<td>32 bytes</td>
</tr>
<tr>
<td>0b0110</td>
<td>64 bytes</td>
</tr>
<tr>
<td>0b0111</td>
<td>128 bytes</td>
</tr>
<tr>
<td>0b1000</td>
<td>256 bytes</td>
</tr>
<tr>
<td>0b1001</td>
<td>512 bytes</td>
</tr>
<tr>
<td>0b1010</td>
<td>1024 bytes</td>
</tr>
<tr>
<td>0b1011</td>
<td>2KB</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.
**Interval, bits [11:8]**

Recommended minimum sampling interval. This provides guidance from the implementer to the smallest minimum sampling interval, N.

<table>
<thead>
<tr>
<th>Interval</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>256</td>
</tr>
<tr>
<td>0b0010</td>
<td>512</td>
</tr>
<tr>
<td>0b0011</td>
<td>768</td>
</tr>
<tr>
<td>0b0100</td>
<td>1,024</td>
</tr>
<tr>
<td>0b0101</td>
<td>1,536</td>
</tr>
<tr>
<td>0b0110</td>
<td>2,048</td>
</tr>
<tr>
<td>0b0111</td>
<td>3,072</td>
</tr>
<tr>
<td>0b1000</td>
<td>4,096</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

**Bits [7:6]**

Reserved, RES0.

**ERnd, bit [5]**

Defines how the random number generator is used in determining the interval between samples, when enabled by PMSIRR_EL1.RND.

<table>
<thead>
<tr>
<th>ERnd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The random number is added at the start of the interval, and the sample is taken and a new interval started when the combined interval expires.</td>
</tr>
<tr>
<td>0b1</td>
<td>The random number is added and the new interval started after the interval programmed in PMSIRR_EL1.INTERVAL expires, and the sample is taken when the random interval expires.</td>
</tr>
</tbody>
</table>

**LDS, bit [4]**

Data source indicator for sampled load instructions

<table>
<thead>
<tr>
<th>LDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Loaded data source not implemented</td>
</tr>
<tr>
<td>0b1</td>
<td>Loaded data source implemented</td>
</tr>
</tbody>
</table>

**ArchInst, bit [3]**

Architectural instruction profiling

<table>
<thead>
<tr>
<th>ArchInst</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Micro-op sampling implemented</td>
</tr>
<tr>
<td>0b1</td>
<td>Architecture instruction sampling implemented</td>
</tr>
</tbody>
</table>

**FL, bit [2]**

Filtering by latency. This bit is RAO.

**FT, bit [1]**

Filtering by operation type. This bit is RAO.

**FE, bit [0]**

Filtering by events. This bit is RAO.
Accessing the PMSIDR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMSIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSIDR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMSIDR_EL1;
  end;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMSIDR_EL1;
  end;
elsif PSTATE.EL == EL3 then
  return PMSIDR_EL1;
The PMSIR_EL1 characteristics are:

**Purpose**

Defines the interval between samples

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSIR_EL1 are **UNDEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

PMSIR_EL1 is a 64-bit register.

**Field descriptions**

The PMSIR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>INTERVAL, bits [31:8]</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>RND, bit [0]</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**INTERVAL, bits [31:8]**

Bits [31:8] of the PMSICR_EL1 interval counter reload value. Software must set this to a non-zero value. If software sets this to zero, an **UNKNOWN** sampling interval is used. Software should set this to a value greater than the minimum indicated by PMSIDR_EL1.Interval

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [7:1]**

Reserved, RES0.

**RND, bit [0]**

Controls randomization of the sampling interval

<table>
<thead>
<tr>
<th>RND</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disable randomization of sampling interval</td>
</tr>
<tr>
<td>0b1</td>
<td>Add (pseudo-)random jitter to sampling interval</td>
</tr>
</tbody>
</table>

The random number generator is not architected.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
Accessing the PMSIRR_EL1

Accesses to this register use the following encodings:

MRS <Xt>, PMSIRR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSIRR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.FGTEn == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x840];
    else
        return PMSIRR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSIRR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return PMSIRR_EL1;

MSR PMSIRR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
  HDFGWTR_EL2.PMSIRR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '1x1' then
    NVMem[0x840] = X[t];
  else
    PMSIRR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMSIRR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  PMSIRR_EL1 = X[t];
The PMSLATFR_EL1 characteristics are:

**Purpose**

Controls sample filtering by latency

**Configuration**

This register is present only when SPE is implemented. Otherwise, direct accesses to PMSLATFR_EL1 are undefined. RW fields in this register reset to architecturally unknown values.

**Attributes**

PMSLATFR_EL1 is a 64-bit register.

**Field descriptions**

The PMSLATFR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:12]**

Reserved, RES0.

**MINLAT, bits [11:0]**

Minimum latency. When PMSFCR_EL1.FL == 1, defines the minimum total latency for filtered operations. Samples with a total latency less than MINLAT will not be recorded.

This field is ignored by the PE when PMSFCR_EL1.FL == 0.

On a Warm reset, this field resets to an architecturally unknown value.

**Accessing the PMSLATFR_EL1**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS</th>
<th>PMSLATFR_EL1</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>op1</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.PMSLATFR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
    MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
    MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return PMSLATFR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
    MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
    MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return PMSLATFR_EL1;
elsif PSTATE.EL == EL3 then
  return PMSLATFR_EL1;

MSR PMSLATFR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.PMSLATFR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
    MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
    MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  PMSLATFR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
    MDCR_EL3.NSPB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
    MDCR_EL3.NSPB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  PMSLATFR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  PMSLATFR_EL1 = X[t];
PMSWINC_EL0, Performance Monitors Software Increment register

The PMSWINC_EL0 characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event 0x00. For more information, see ‘SW_INCR’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D5.

Configuration

AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSWINC[31:0].

AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to External register PMSWINC_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMSWINC_EL0 are UNDEFINED.

Attributes

PMSWINC_EL0 is a 64-bit register.

Field descriptions

The PMSWINC_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:31]</th>
<th>RES0</th>
<th>P&lt;n&gt;, bit [n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>RES0</td>
<td>P&lt;n&gt;, bit [n]</td>
</tr>
</tbody>
</table>

Bits [63:31]

Reserved, RES0.

P<n>, bit [n], for n = 0 to 30

Event counter software increment bit for PMEVCNTR<n>_EL0.

If N is less than 31, then bits [30:N] are WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN. Otherwise, N is the value in PMCR_EL0.N.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action. The write to this bit is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>If PMEVCNTR&lt;n&gt;_EL0 is enabled and configured to count the software increment event, increments PMEVCNTR&lt;n&gt;_EL0 by 1. If PMEVCNTR&lt;n&gt;_EL0 is disabled, or not configured to count the software increment event, the write to this bit is ignored.</td>
</tr>
</tbody>
</table>

Accessing the PMSWINC_EL0

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<SW,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSWINC_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMSWINC_EL0 = X[t];
  endif
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
      SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSWINC_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMSWINC_EL0 = X[t];
  endif
else
  if PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMSWINC_EL0 = X[t];
    endif
  elsif PSTATE.EL == EL3 then
    PMSWINC_EL0 = X[t];
  endif
end if
PMUSERENR_EL0, Performance Monitors User Enable Register

The PMUSERENR_EL0 characteristics are:

Purpose

Enables or disables EL0 access to the Performance Monitors.

Configuration

AArch64 System register PMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMUSERENR[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMUSERENR_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

PMUSERENR_EL0 is a 64-bit register.

Field descriptions

The PMUSERENR_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>ER, bit [3]</td>
</tr>
<tr>
<td>61</td>
<td>Event counter Read. Traps EL0 access to event counters to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.</td>
</tr>
<tr>
<td>60</td>
<td>In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.</td>
</tr>
<tr>
<td>59</td>
<td>In AArch64 state, trapped accesses are reported using EC syndrome value 0x03.</td>
</tr>
</tbody>
</table>

ER, bit [3]

Event counter Read. Traps EL0 access to event counters to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>ER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 using AArch64: EL0 reads of the PMXEVCNTR_EL0 and PMEVCTR&lt;n&gt;.EL0, and EL0 read/write accesses to the PMSEL_R_EL0, are trapped if PMUSERENR_EL0.EN is also 0. EL0 using AArch32: EL0 reads of the PMXEVCNTR and PMEVCTR&lt;n&gt;, and EL0 read/write accesses to the PMSEL_R, are trapped if PMUSERENR_EL0.EN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overrides PMUSERENR_EL0.EN and enables RO access to PMXEVCTR_EL0 and PMEVCTR&lt;n&gt;.EL0, and RW access to PMSEL_R_EL0 and PMSEL_R at EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.
CR, bit [2]

Cycle counter Read. Traps EL0 access to cycle counter reads to EL1, or to EL2 when it is implemented and enabled for
the current Security state and HCR_EL2.TGE is 1.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.

In AArch32 state, trapped MRC accesses are reported using EC syndrome value 0x03, trapped MRRC accesses are
reported using EC syndrome value 0x04.

<table>
<thead>
<tr>
<th>CR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 using AArch64: EL0 read accesses to the PMCCNTR_EL0 are trapped if PMUSERENR_EL0.EN is also 0. EL0 using AArch32: EL0 read accesses to the PMCCNTR are trapped if PMUSERENR_EL0.EN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overrides PMUSERENR_EL0.EN and enables access to PMCCNTR_EL0 and PMCCNTR at EL0.</td>
</tr>
</tbody>
</table>

SW, bit [1]

Traps Software Increment writes to EL1, or to EL2 when it is implemented and enabled for the current Security state
and HCR_EL2.TGE is 1.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.

In AArch32 state, trapped accesses are reported using EC syndrome value 0x03.

<table>
<thead>
<tr>
<th>SW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 using AArch64: EL0 writes to the PMSWINC_EL0 are trapped if PMUSERENR_EL0.EN is also 0. EL0 using AArch32: EL0 writes to the PMSWINC are trapped if PMUSERENR_EL0.EN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overrides PMUSERENR_EL0.EN and enables access to PMSWINC_EL0 and PMSWINC at EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Traps EL0 accesses to the Performance Monitor registers to EL1, or to EL2 when it is implemented and enabled for
the current Security state and HCR_EL2.TGE is 1, from both Execution states as follows:

- In AArch64 state, MRS or MSR accesses to the following registers are reported using EC syndrome value 0x18:
  - PMCR_EL0, PMOVSCMR_EL0, PMSELR_EL0, PMCEID0_EL0, PMCEID1_EL0, PMCCNTR_EL0, PMXEVTPR_EL0, PMXEVNTR_EL0, PMUSERENR_EL0, PMCNTENSET_EL0, PMCNTENCLR_EL0, PMOVSSET_EL0, PMXEVTYPER<n>_EL0, PMEVTYPER<n>_EL0, PMCCFILTER_EL0.
  - PMSWINC_EL0, MSR accesses only.
- In AArch32 state, MRC or MCR accesses to the following registers are reported using EC syndrome value 0x03, MRRC or MCRR accesses are trapped to EL2 and reported using EC syndrome value 0x04:
  - PMCR, PMOVSR, PMSELR, PMCEID0, PMCEID1, PMCCNTR, PMXEVTPR, PMXEVNTR, PMUSERENR, PMCNTENSET, PMCNTENCLR, PMOVSSET, PMEVNTR<n>, PMEVTYPER<n>, PMCCFILTER.
  - PMSWINC, MCR accesses only.
- In AArch32 state, MRRC or MCRR accesses to PMCCNTR are reported using EC syndrome value 0x04.
- If ARMv8.4-PMU is implemented, in AArch64 state, PMMIR_EL1 and in AArch32 state, PMMIR.
- If ARMv8.1-PMU is implemented, in AArch32 state, PMCEID2, and PMCEID3.
### PMUSERENR_EL0, Performance Monitors User Enable Register

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>While at EL0, Accesses to the specified registers at EL0 are trapped, unless overridden by one of PMUSERENR_EL0.{ER, CR, SW}.</td>
</tr>
<tr>
<td>0b1</td>
<td>While at EL0, software can access all of the specified registers.</td>
</tr>
</tbody>
</table>

#### Note

The EL0 access is trapped only if the corresponding EL1 access is permitted. If PMUSERENR_EL0.EN is 0, write access to PMSWINC_EL0 and PMSWINC from EL0 are trapped, but read access is UNDEFINED.

The affected registers do not include PMINTENSET_EL1, PMINTENCLR_EL1, PMINTENSET and PMINTENCLR.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### Accessing the PMUSERENR_EL0

Accesses to this register use the following encodings:

**MRS <Xt>, PMUSERENR_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```assembly
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && !HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1' && HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMUSERENR_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
        SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMUSERENR_EL0;
        elsif PSTATE.EL == EL2 then
            if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return PMUSERENR_EL0;
        elsif PSTATE.EL == EL3 then
            return PMUSERENR_EL0;

MRS PMUSERENR_EL0, <Xt>```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUSERENR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
    elsif PSTATE.EL == EL3 then
        PMUSERENR_EL0 = X[t];
else
    PMUSERENR_EL0 = X[t];
else
    PMUSERENR_EL0 = X[t];
PMXEVCNTR_EL0, Performance Monitors Selected Event Count Register

The PMXEVCNTR_EL0 characteristics are:

**Purpose**

Reads or writes the value of the selected event counter, \( \text{PMEVCNTR}<n> \_\text{EL0} \). \( \text{PMSELR}_\text{EL0} \). SEL determines which event counter is selected.

**Configuration**

AArch64 System register PMXEVCNTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVCNTR[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMXEVCNTR_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMXEVCNTR_EL0 is a 64-bit register.

**Field descriptions**

The PMXEVCNTR_EL0 bit assignments are:

When ARMv8.5-PMU is implemented:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>PMEVCNTR&lt;n&gt;</td>
<td>Value of the selected event counter, ( \text{PMEVCNTR}&lt;n&gt; _\text{EL0} ), where n is the value stored in ( \text{PMSELR}_\text{EL0} ). SEL.</td>
</tr>
<tr>
<td>31</td>
<td>PMEVCNTR&lt;n&gt;</td>
<td>Value of the selected event counter, ( \text{PMEVCNTR}&lt;n&gt; _\text{EL0} ), where n is the value stored in ( \text{PMSELR}_\text{EL0} ). SEL.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>PMEVCNTR&lt;n&gt;</td>
<td>Value of the selected event counter, ( \text{PMEVCNTR}&lt;n&gt; _\text{EL0} ), where n is the value stored in ( \text{PMSELR}_\text{EL0} ). SEL.</td>
</tr>
</tbody>
</table>
On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

# Accessing the PMXEVCNTR_EL0

If ARMv8.6-FGT is implemented, and EL2 is implemented and enabled in the current Security state, for reads and writes at EL1 and permitted reads and writes at EL0:

- If **PMSELR_EL0**.SEL is an unimplemented event counter, the access is **UNDEFINED**.
- Otherwise, if **PMSELR_EL0**.SEL is greater than or equal to **MDCR_EL2**.HPMN, the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:

If **PMSELR_EL0**.SEL is greater than or equal to the number of accessible counters then reads and writes of **PMXEVCNTR_EL0** are **CONSTRAINED UNPREDICTABLE**, and the following behaviors are permitted:

- Accesses to the register are **UNDEFINED**.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if **PMSELR_EL0**.SEL has an **UNKNOWN** value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and **PMSELR_EL0**.SEL is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

## Note

In EL0, an access is permitted if it is enabled by **PMUSERENR_EL0**.{ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, **MDCR_EL2**.HPMN identifies the number of accessible counters. Otherwise, the number of accessible counters is the number of implemented counters. See **MDCR_EL2**.HPMN for more details.

Accesses to this register use the following encodings:

**MRS <Xt>, PMXEVCNTR_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMXEVCNTR_EL0;
    elsif PSTATE_EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVCNTR_EL0;
        elsif PSTATE_EL == EL2 then
            if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return PMXEVCNTR_EL0;
        elsif PSTATE_EL == EL3 then
            return PMXEVCNTR_EL0;
        MSR PMXEVCNTR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMXEVCNTR_EL0 = X[t];
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMXEVCNTR_EL0 = X[t];
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMXEVCNTR_EL0 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      PMXEVCNTR_EL0 = X[t];
  elsif PSTATE.EL == EL3 then
    PMXEVCNTR_EL0 = X[t];

PMXEVTYPER_EL0, Performance Monitors Selected Event Type Register

The PMXEVTYPER_EL0 characteristics are:

**Purpose**

When PMSELR_EL0 SEL selects an event counter, this accesses a PMEVTYPER<n>_EL0 register. When PMSELR_EL0SEL selects the cycle counter, this accesses PMCCFILTR_EL0.

**Configuration**

AArch64 System register PMXEVTYPER_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMXEVTYPER[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMXEVTYPER_EL0 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMXEVTYPER_EL0 is a 64-bit register.

**Field descriptions**

The PMXEVTYPER_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Event type register</td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**Event type register or PMCCFILTR_EL0, bits [31:0]**

When PMSELR_EL0 SEL == 31, this register accesses PMCCFILTR_EL0.

Otherwise, this register accesses PMEVTYPER<n>_EL0 where n is the value in PMSELR_EL0.SEL.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMXEVTYPER_EL0**

If ARMv8.6-FGT is implemented, and EL2 is implemented and enabled in the current Security state, for reads and writes at EL1 and permitted reads and writes at EL0:

- If PMSELR_EL0.SEL is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, if PMSELR_EL0.SEL is not 31 and greater than or equal to MDCR_EL2.HPMN, the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:
If PMSEL_EL0.SEL is not 31 and greater than or equal to the number of accessible counters then reads and writes of PMXEVTYPER_EL0 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSEL_EL0.SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if PMSEL_EL0.SEL is 31.
- If EL2 is implemented and enabled in the current Security state, PMSEL_EL0 is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

**Note**

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0.EN.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, MDCR_EL2.HPMN identifies the number of accessible counters.
Otherwise, the number of accessible counters is the number of implemented counters. See MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings:

**MRS <Xt>, PMXEVTYPER_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end;
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMXEVTYPER_EL0;
  end;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMXEVTYPER_EL0;
  end;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return PMXEVTYPER_EL0;
  end;
elsif PSTATE.EL == EL3 then
  return PMXEVTYPER_EL0;
endif;

MSR PMXEVTYPER_EL0, <Xt>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    endif
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMXEVTYPER_EL0 = X[t];
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMXEVTYPER_EL0 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    PMXEVTYPER_EL0 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  PMXEVTYPER_EL0 = X[t];
end
REVIDR_EL1, Revision ID Register

The REVIDR_EL1 characteristics are:

**Purpose**

Provides implementation-specific minor revision information.

**Configuration**

AArch64 System register REVIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register MIDR_EL1.

If REVIDR_EL1 has the same value as MIDR_EL1, then its contents have no significance.

**Attributes**

REVIDR_EL1 is a 64-bit register.

**Field descriptions**

The REVIDR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**IMPLEMENTATION DEFINED, bits [63:0]**

**IMPLEMENTATION DEFINED.**

**Accessing the REVIDR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, REVIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if IsFeatureImplemented("ARMv8.4-IDST") then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRTR_EL2.REVIDR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return REVIDR_EL1;
elsif PSTATE.EL == EL2 then
  return REVIDR_EL1;
else
  return REVIDR_EL1;
RGSR_EL1, Random Allocation Tag Seed Register.

The RGSR_EL1 characteristics are:

**Purpose**

Random Allocation Tag Seed Register.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to RGSR_EL1 are **UNDEFINED**.

When GCR_EL1.RRND==0b1, the value of RGSR_EL1 is **UNKNOWN**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

RGSR_EL1 is a 64-bit register.

**Field descriptions**

The RGSR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| **RES0** | **SEED** | **RES0** | **TAG** |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:24]**

Reserved, **RES0**.

**SEED, bits [23:8]**

Seed register used for generating values returned by RandomAllocationTag().

This field resets to an architecturally **UNKNOWN** value.

**Bits [7:4]**

Reserved, **RES0**.

**TAG, bits [3:0]**

Tag generated by the most recent IRG instruction.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the RGSR_EL1**

Accesses to this register use the following encodings:
MRS <Xt>, RGSR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return RGSR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return RGSR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return RGSR_EL1;

MSR RGSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    RGSR_EL1 = X[t];
RMR_EL1, Reset Management Register (EL1)

The RMR_EL1 characteristics are:

Purpose

If EL1 is the highest implemented Exception level and this register is implemented:

- A write to the register at EL1 can request a Warm reset.
- If EL1 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register RMR[31:0] when IsHighestEL(EL1).

Only implemented if EL1 is the highest implemented Exception level. In this case:

- If EL1 can use AArch32 and AArch64 then this register must be implemented.
- If EL1 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

RMR_EL1 is a 64-bit register.

Field descriptions

The RMR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 | RR | AA64 |

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

This field resets to 0.

AA64, bit [0]

When EL1 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

<table>
<thead>
<tr>
<th>AA64</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>AArch64.</td>
</tr>
</tbody>
</table>

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.
If EL1 cannot use AArch32 this bit is RAO/WI.
When implemented as a RW field, this field resets to 1 on a Cold reset.

**Accessing the RMR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, RMR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL1 && IsHighestEL(EL1) then
    return RMR_EL1;
else
    UNDEFINED;
```

MSR RMR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL1 && IsHighestEL(EL1) then
    RMR_EL1 = X[t];
else
    UNDEFINED;
```
The RMR_EL2 characteristics are:

**Purpose**

If EL2 is the highest implemented Exception level and this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

**Configuration**

AArch64 System register RMR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HRMR[31:0] when IsHighestEL(EL2).

Only implemented if EL2 is the highest implemented Exception level. In this case:

- If EL2 can use AArch32 and AArch64 then this register must be implemented.
- If EL2 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

RMR_EL2 is a 64-bit register.

**Field descriptions**

The RMR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-2</td>
<td>Reserved, RES0.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reset Request. Setting this bit to 1 requests a Warm reset.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>When EL2 can use AArch32, determines which Execution state the PE boots into after a Warm reset:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0b0</td>
<td>AArch32.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0b1</td>
<td>AArch64.</td>
<td></td>
</tr>
</tbody>
</table>
On coming out of the Warm reset, execution starts at the implementation-defined reset vector address of the specified Execution state.

If EL2 cannot use AArch32 this bit is RAO/WI.

When implemented as a RW field, this field resets to 1 on a Cold reset.

**Accessing the RMR_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, RMR_EL2**

```markdown
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

```python
if PSTATE.EL == EL1 && EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    return RMR_EL2;
else
    UNDEFINED;
```

**MSR RMR_EL2, <Xt>**

```markdown
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

```python
if PSTATE.EL == EL1 && EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    RMR_EL2 = X[t];
else
    UNDEFINED;
```
RMR_EL3, Reset Management Register (EL3)

The RMR_EL3 characteristics are:

**Purpose**

If EL3 is the implemented and this register is implemented:

- A write to the register at EL3 can request a Warm reset.
- If EL3 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

**Configuration**

AArch64 System register RMR_EL3 bits [31:0] are architecturally mapped to AArch32 System register RMR[31:0] when HaveEL(EL3).

When EL3 is implemented:

- If EL3 can use AArch32 and AArch64, then this register must be implemented.
- If EL3 cannot use AArch32, then it is IMPLEMENTATION DEFINED whether the register is implemented.

Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

RMR_EL3 is a 64-bit register.

**Field descriptions**

The RMR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ | \_ |

**Bits [63:2]**

Reserved, RES0.

**RR, bit [1]**

Reset Request. Setting this bit to 1 requests a Warm reset.

This field resets to 0.

**AA64, bit [0]**

When EL3 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

<table>
<thead>
<tr>
<th>AA64</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>AArch64.</td>
</tr>
</tbody>
</table>
On coming out of the Warm reset, execution starts at the implementation defined reset vector address of the specified execution state.

If EL3 cannot use AArch32 this bit is RAO/WI.

When implemented as a RW field, this field resets to 1 on a Cold reset.

**Accessing the RMR_EL3**

Accesses to this register use the following encodings:

**MRS <Xt>, RMR_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL3 && IsHighestEL(EL3) then
    return RMR_EL3;
else
    UNDEFINED;

**MSR RMR_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL3 && IsHighestEL(EL3) then
    RMR_EL3 = X[t];
else
    UNDEFINED;
RNDR, Random Number

The RNDR characteristics are:

**Purpose**

Random Number. Returns a 64-bit random number which is reseeded from the True Random Number source at an IMPLEMENTATION DEFINED rate.

If the hardware returns a genuine random number, PSTATE.NZCV is set to 0b0000.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to 0b0100 and the data value returned in UNKNOWN.

RNDR is a read-only register.

**Configuration**

This register is present only when ARMv8.5-RNG is implemented. Otherwise, direct accesses to RNDR are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

RNDR is a 64-bit register.

**Field descriptions**

The RNDR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

RNDR, bits [63:0]

Random Number. Returns a 64-bit Random Number which is reseeded from the True Random Number source at an IMPLEMENTATION DEFINED rate.

This field resets to an architecturally UNKNOWN value.

**Accessing the RNDR**

Accesses to this register use the following encodings:

\[
\text{MRS} \ <Xt>, \ \text{RNDR}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0010</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    return RNDR;
elsif PSTATE.EL == EL1 then
    return RNDR;
elsif PSTATE.EL == EL2 then
    return RNDR;
elsif PSTATE.EL == EL3 then
    return RNDR;
RNDRRS, Reseeded Random Number

The RNDRRS characteristics are:

**Purpose**

Reseeded Random Number. Returns a 64-bit random number which is reseeded from the True Random Number source at an IMPLEMENTATION DEFINED rate.

If the hardware returns a genuine random number, PSTATE.NZCV is set to 0b0000.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to 0b0100 and the data value returned is UNKNOWN.

RNDRRS is a read-only register.

**Configuration**

This register is present only when ARMv8.5-RNG is implemented. Otherwise, direct accesses to RNDRRS are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

RNDRRS is a 64-bit register.

**Field descriptions**

The RNDRRS bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RNDRRS |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

RNDRRS, bits [63:0]

Reseeded Random Number. Returns a 64-bit Random Number which is reseeded from the True Random Number source immediately before this read.

This field resets to an architecturally UNKNOWN value.

**Accessing the RNDRRS**

Accesses to this register use the following encodings:

MRS <Xt>, RNDRRS

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0010</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    return RNDRRS;
elif PSTATE.EL == EL1 then
    return RNDRRS;
elif PSTATE.EL == EL2 then
    return RNDRRS;
elif PSTATE.EL == EL3 then
    return RNDRRS;
RVBAR_EL1, Reset Vector Base Address Register (if EL2 and EL3 not implemented)

The RVBAR_EL1 characteristics are:

**Purpose**

If EL1 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

**Configuration**

Only implemented if the highest Exception level implemented is EL1.

**Attributes**

RVBAR_EL1 is a 64-bit register.

**Field descriptions**

The RVBAR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Reset Address. The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

**Accessing the RVBAR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, RVBAR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 && IsHighestEL(EL1) then
  return RVBAR_EL1;
else
  UNDEFINED;
RVBAR_EL2, Reset Vector Base Address Register (if EL3 not implemented)

The RVBAR_EL2 characteristics are:

**Purpose**

If EL2 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

**Configuration**

Only implemented if the highest Exception level implemented is EL2.

This register has no effect if EL2 is not enabled in the current Security state.

**Attributes**

RVBAR_EL2 is a 64-bit register.

**Field descriptions**

The RVBAR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reset Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Reset Address. The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

**Accessing the RVBAR_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, RVBAR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 && EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    return RVBAR_EL2;
else
    UNDEFINED;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
RVBAR_EL2, Reset Vector Base Address Register (if EL3 not implemented)
RVBAR_EL3, Reset Vector Base Address Register (if EL3 implemented)

The RVBAR_EL3 characteristics are:

**Purpose**

If EL3 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to RVBAR_EL3 are UNDEFINED.

Only implemented if the highest Exception level implemented is EL3.

**Attributes**

RVBAR_EL3 is a 64-bit register.

**Field descriptions**

The RVBAR_EL3 bit assignments are:

| Bits  | 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|       |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [63:0]**

Reset Address. The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

**Accessing the RVBAR_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, RVBAR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL3 && IsHighestEL(EL3) then
  return RVBAR_EL3;
else
  UNDEFINED;
S1_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED maintenance instructions

The S1_<op1>_<Cn>_<Cm>_<op2> characteristics are:

Purpose

This area of the System instruction encoding space is reserved for IMPLEMENTATION DEFINED System instructions.

Configuration

Attributes

S1_<op1>_<Cn>_<Cm>_<op2> is a 64-bit System instruction.

Field descriptions

The S1_<op1>_<Cn>_<Cm>_<op2> input value bit assignments are:

### IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Executing the S1_<op1>_<Cn>_<Cm>_<op2> instruction

Accesses to this instruction use the following encodings:

SYS #<op1>, <Cn>, <Cm>, #<op2>{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>op1[2:0]</td>
<td>0b1x11</td>
<td>Cm[3:0]</td>
<td>op2[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.TIDCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    IMPLEMENTATION_DEFINED ""
  else
    IMPLEMENTATION_DEFINED "";

SYSL <Xt>, #<op1>, <Cn>, <Cm>, #<op2>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>op1[2:0]</td>
<td>0b1x11</td>
<td>Cm[3:0]</td>
<td>op2[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TIDCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        IMPLEMENTATION_DEFINED "";
else
    IMPLEMENTATION_DEFINED "";
S3_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED registers

The S3_<op1>_<Cn>_<Cm>_<op2> characteristics are:

**Purpose**

This area of the instruction set space is reserved for IMPLEMENTATION DEFINED registers.

**Configuration**

**Attributes**

S3_<op1>_<Cn>_<Cm>_<op2> is a 64-bit register.

**Field descriptions**

The S3_<op1>_<Cn>_<Cm>_<op2> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>60</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>59</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>58</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>57</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>56</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>55</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>54</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>53</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>52</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>51</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>50</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>49</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>48</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>47</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>46</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>45</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>44</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>43</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>42</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>41</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>40</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>39</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>38</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>37</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>36</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>35</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>34</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>33</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:0]**

IMPLENENTATION DEFINED.

**Accessing the S3_<op1>_<Cn>_<Cm>_<op2>**

Accesses to this register use the following encodings:

MRS <Xt>, S3_<op1>_C<Cn>_C<Cm>_<op2>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>op1[2:0]</td>
<td>0b1x11</td>
<td>Cm[3:0]</td>
<td>op2[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 then
  if EL2Enabled() & & HCR_EL2.TIDCP == '1' then
    AArch64.SystemAccessTrap(El2, 0x18);
  else
    IMPLEMENTATION_DEFINED "";
else
  IMPLEMENTATION_DEFINED "";

MSR S3_<op1>_C<Cn>_C<Cm>_<op2>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>op1[2:0]</td>
<td>0b1x11</td>
<td>Cm[3:0]</td>
<td>op2[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.TIDCP == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    IMPLEMENTATION_DEFINED ""
else
  IMPLEMENTATION_DEFINED "";
The SCR_EL3 characteristics are:

**Purpose**

Defines the configuration of the current Security state. It specifies:

- The Security state of EL0, EL1, and EL2. The Security state is either Secure or Non-secure.
- The Execution state at lower Exception levels.
- Whether IRQ, FIQ, SError interrupts, and External abort exceptions are taken to EL3.
- Whether various operations are trapped to EL3.

**Configuration**

AArch64 System register SCR_EL3 bits [31:0] can be mapped to AArch32 System register SCR[31:0], but this is not architecturally mandated.

This register is present only when HaveEL(EL3). Otherwise, direct accesses to SCR_EL3 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SCR_EL3 is a 64-bit register.

**Field descriptions**

The SCR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | TWEDEn | TWEDEn | AMVOFFEN | EN | ATAEn | SCXTr | RES0 | FIEN | NMEAEn | EASEEn | EEL2API | APKt | TERR | TLOR | TWETW | STR | RW | SIF | HCE | SMD | RES0 | RES0 | RES0 | RES0 | RES0 |

Bits [63:36]

Reserved, RES0.

**AMVOFFEN, bit [35]**

When ARMv8.6-AMU is implemented:

Activity Monitors Virtual Offsets Enable.

<table>
<thead>
<tr>
<th>AMVOFFEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>アクセスes to AMEVCNTVOFF0&lt;\textless n\textgreater &gt; EL2 and AMEVCNTVOFF1&lt;\textless n\textgreater &gt; EL2 at EL2 are trapped to EL3. Indirect reads of the virtual offset registers are zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>アクセスes to AMEVCNTVOFF0&lt;\textless n\textgreater &gt; EL2 and AMEVCNTVOFF1&lt;\textless n\textgreater &gt; EL2 are not affected by this field.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.
TME, bit [34]

When TME is implemented:

Enables access to the TSTART, TCOMMIT, TTEST and TCANCEL instructions at EL0, EL1 and EL2.

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0, EL1 and EL2 accesses to TSTART, TCOMMIT, TTEST and TCANCEL instructions are UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instruction to be UNDEFINED.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [33:30]

When ARMv8.6-TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCR_EL3.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by SCR_EL3.TWE as \(2^{(\text{TWEDEL} + 8)}\) cycles.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [29]

When ARMv8.6-TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by SCR_EL3.TWE.

<table>
<thead>
<tr>
<th>TWEDEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The delay for taking a WFE trap is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>The delay for taking a WFE trap is at least the number of cycles defined in SCR_EL3.TWEDEL.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECVEn, bit [28]

When ARMv8.6-ECV is implemented:

ECV Enable. Enables access to the CNTPOFF_EL2 register.

<table>
<thead>
<tr>
<th>ECVEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL2 accesses to CNTPOFF_EL2 are trapped to EL3, and the value of CNTPOFF_EL2 is treated as 0 for all purposes other than direct reads or writes to the register from EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 accesses to CNTPOFF_EL2 are not trapped to EL3 by this mechanism.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

FGTEn, bit [27]

When ARMv8.6-FGT is implemented:

Enables access to the Fine-Grained Traps registers: HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGTR_EL2 and HFGWTR_EL2.

<table>
<thead>
<tr>
<th>FGTEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL2 Accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGTR_EL2 and HFGWTR_EL2 registers are trapped to EL3, and those registers behave as if all bits are set to 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 Accesses to HAFGRTR_EL2, HDFGRTR_EL2, HDFGWTR_EL2, HFGTR_EL2 and HFGWTR_EL2 registers are not trapped to EL3 by this mechanism.</td>
</tr>
</tbody>
</table>

Traps caused by accesses to the fine-grained trap registers are reported using the EC code of 0x18 and its associated ISS.

Otherwise:

Reserved, RES0.

ATA, bit [26]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access. Controls access at EL2, EL1 and EL0 to Allocation Tags.

When access is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.
- MRS and MSR instructions at EL1 and EL2 using GCR_EL1, RGSR_EL1, TFSR_EL1, TFSR_EL2 or TFSRE0_EL1 that are not UNDEFINED or trapped to a lower Exception level are trapped to EL3.
- MRS and MSR instructions at EL2 using TFSR_EL2 that are not UNDEFINED are trapped to EL3.

<table>
<thead>
<tr>
<th>ATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access is prevented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access is not prevented.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [25]
When ARMv8.0-CSV2 is implemented:

Enable access to the SCXTNUM_EL2, SCXTNUM_EL1, and SCXTNUM_EL0 registers. The defined values are:

<table>
<thead>
<tr>
<th>EnSCXT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL2, EL1 and EL0 access to SCXTNUM_EL0, EL2 and EL1 access to SCXTNUM_EL1, EL2 access to SCXTNUM_EL2 registers are disabled by this mechanism, causing an exception to EL3, and the values of these registers to be treated as 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause accesses to SCXTNUM_EL0, SCXTNUM_EL1, SCXTNUM_EL2 to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:22]

Reserved, RES0.

FIEN, bit [21]

When ARMv8.4-RAS is implemented:

Fault Injection enable. Trap accesses to the RAS ERX* registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1 from EL1 and EL2 to EL3, reported using EC syndrome value 0x18.

<table>
<thead>
<tr>
<th>FIEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented, the Effective value of SCR_EL3.FIEN is 0b1.

If ERRIDR_EL1.NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [20]

When ARMv8.4-DFE is implemented:

Non-maskable External Aborts. When SCR_EL3.EA == 1, controls whether PSTATE.A masks SError interrupts at EL3.

<table>
<thead>
<tr>
<th>NMEA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If SCR_EL3.EA == 1, asserted SError interrupts are not taken at EL3 if PSTATE.A == 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>If SCR_EL3.EA == 1, asserted SError interrupts are taken at EL3 regardless of the value of PSTATE.A.</td>
</tr>
</tbody>
</table>

When SCR_EL3.EA == 0:

- Asserted SError interrupts are not taken at EL3 regardless of the value of PSTATE.A and this field.
- This field is ignored and its Effective value is 0.
This field resets to 0.

Otherwise:

Reserved, RES0.

**EASE, bit [19]**

When ARMv8.4-DFE is implemented:

External aborts to SError interrupt vector.

<table>
<thead>
<tr>
<th>EASE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError interrupt vector offset from VBAR_EL3.</td>
</tr>
</tbody>
</table>

This field resets to 0.

Otherwise:

Reserved, RES0.

**EEL2, bit [18]**

When ARMv8.4-SecEL2 is implemented:

Secure EL2 Enable.

<table>
<thead>
<tr>
<th>EEL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All behaviors associated with Secure EL2 are disabled. All registers, including timer registers, defined by ARMv8.4-SecEL2 are UNDEFINED, and those timers are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>All behaviors associated with Secure EL2 are enabled.</td>
</tr>
</tbody>
</table>

When the value of this bit is 1, then:

- When SCR_EL3.NS == 0, the SCR_EL3.RW bit is treated as 1 for all purposes other than reading or writing the register.

- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2, using the EC value of ESR_EL2,EC== 0x3:
  - A read or write of the SCR.
  - A read or write of the NSACR.
  - A read or write of the MVBAR.
  - A read or write of the SDCR.
  - Execution of an ATS12NSO** instruction.

- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using the EC value of ESR_EL2,EC== 0x0:
  - Execution of an SRS instruction that uses R13_mon.
  - Execution of an MRS (Banked register) or MSR (Banked register) instruction that would access SPSR_mon, R13_mon, or R14_mon.

**Note**

If the Effective value of SCR_EL3.EEL2 is 0, then these operations executed in Secure EL1 using AArch32 are trapped to EL3.
In a Secure only implementation that does not implement EL3 but implements EL2, behaves as if SCR_EL3.EEL2 == 1.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [17]

When ARMv8.4-SecEL2 is implemented and ARMv8.3-PAuth is implemented:

Controls the use of the following instructions related to Pointer Authentication. Traps are reported using EC syndrome value 0x09:

- PACGA, which is always enabled.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BRBA, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETTA, ERETTB, LDRAA and LDRAB when:
  - In EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M> == 1.
  - In EL0, when HCR_EL2.TGE==1 and HCR_EL2.E2H==1, and the associated SCTLR_EL2.En<N><M> == 1.
  - In EL1, when the associated SCTLR_EL1.En<N><M> == 1.
  - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.

<table>
<thead>
<tr>
<th>API</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

An instruction is trapped only if Pointer Authentication is enabled for that instruction, for more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Note

If ARMv8.3-PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

This field resets to an architecturally UNKNOWN value.

When ARMv8.4-SecEL2 is not implemented and ARMv8.3-PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BRBA, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETTA, ERETTB, LDRAA and LDRAB when:
  - In Non-secure EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated SCTLR_EL1.En<N><M> == 1.
  - In Non-secure EL0, when HCR_EL2.TGE==1 and HCR_EL2.E2H==1, and the associated SCTLR_EL2.En<N><M> == 1.
  - In Secure EL0, when the associated SCTLR_EL2.En<N><M> == 1.
  - In Secure or Non-secure EL1, when the associated SCTLR_EL1.En<N><M> == 1.
  - In EL2, when the associated SCTLR_EL2.En<N><M> == 1.
API

<table>
<thead>
<tr>
<th>API</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the \texttt{HCR_EL2.API} bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

**Note**

If ARMv8.3-PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

This field resets to an architecturally \texttt{UNKNOWN} value.


date

Otherwise:

Reserved, \texttt{RES0}.

**APK, bit [16]**

\textbf{When ARMv8.3-PAuth is implemented:}

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers, using EC syndrome value 0x18, from EL1 or EL2 to EL3 unless they are trapped to EL2 as a result of the \texttt{HCR\_EL2.APK} bit or other traps:

- \texttt{APIAKeyLo\_EL1}, \texttt{APIAKeyHi\_EL1}, \texttt{APIBKeyLo\_EL1}, \texttt{APIBKeyHi\_EL1}.
- \texttt{APDAKeyLo\_EL1}, \texttt{APDAKeyHi\_EL1}, \texttt{APDBKeyLo\_EL1}, \texttt{APDBKeyHi\_EL1}.
- \texttt{APGAKeyLo\_EL1}, \texttt{APGAKeyHi\_EL1}.

<table>
<thead>
<tr>
<th>APK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to the registers holding “key” values for pointer authentication from EL1 or EL2 are trapped to EL3 unless they are trapped to EL2 as a result of the \texttt{HCR_EL2.APK} bit or other traps.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

If ARMv8.3-PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

This field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

**TERR, bit [15]**

\textbf{When RAS is implemented:}

Trap Error record accesses. Accesses to the RAS ERR and RAS ERX registers from EL1 and EL2 to EL3 are trapped as follows:

- If EL1 is using AArch64, \texttt{ERRIDR\_EL1}, \texttt{ERRSEL\_EL1}, \texttt{ERXADDR\_EL1}, \texttt{ERXCTLR\_EL1}, \texttt{ERXFR\_EL1}, \texttt{ERXMISC0\_EL1}, \texttt{ERXMISC1\_EL1}, and \texttt{ERXSTATUS\_EL1}, are trapped and reported using EC syndrome value 0x18.
If ARMv8.4-RAS is implemented, ERXMISC2_EL1 and ERXMISC3_EL1 are trapped and reported using EC syndrome value 0x18.

If EL1 is using AArch32, accesses by MCR or MRC to the following registers are reported using EC syndrome value 0x03, accesses by MCRR or MRRC are reported using EC syndrome value 0x04:

- ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTRL, ERXCTRL2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS.

If ARMv8.4-RAS is implemented, accesses by MCR or MRC to the following registers are reported using EC syndrome value 0x03, accesses by MCRR or MRRC are reported using EC syndrome value 0x04:

- ERXMISC4, ERXMISC5, ERXMISC6, and ERXMISC7.

<table>
<thead>
<tr>
<th>TER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped HCR_EL2.TLOR.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Reserved: RES0.

TLOR, bit [14]

When ARMv8.1-LOR is implemented:

Trap LOR registers. Traps accesses to the LORSA_EL1, LOREFA_EL1, LORN_EL1, LORC_EL1, and LORID_EL1 registers from EL1 and EL2 to EL3, unless the access has been trapped to EL2.

<table>
<thead>
<tr>
<th>TLOR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped HCR_EL2.TLOR.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Reserved: RES0.

TWE, bit [13]

Traps EL2, EL1, and EL0 execution of WFE instructions to EL3, from both Security states and both Execution states, reported using EC syndrome value 0x01.

<table>
<thead>
<tr>
<th>TWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFE instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE, HCR.TWE, SCTLR_EL1.nTWE, SCTLR_EL2.nTWE, or HCR_EL2.TWE.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or
WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile. This field resets to an architecturally UNKNOWN value.

**TWI, bit [12]**

Traps EL2, EL1, and EL0 execution of WFI instructions to EL3, from both Security states and both Execution states, reported using EC syndrome value 0x01.

<table>
<thead>
<tr>
<th>TWI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFI instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI, HCR.TWI, SCTLR_EL1.nTWI, SCTLR_EL2.nTWI, or HCR_EL2.TWI.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile. This field resets to an architecturally UNKNOWN value.

**ST, bit [11]**

Traps Secure EL1 accesses to the Counter-timer Physical Secure timer registers to EL3, from AArch64 state only, reported using EC syndrome value 0x18.

<table>
<thead>
<tr>
<th>ST</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL1 using AArch64 accesses to the CNTPS.TVAL.EL1, CNTPS.CTL.EL1, and CNTPS.CVAL.EL1 are trapped to EL3 when Secure EL2 is disabled. If Secure EL2 is enabled, the behavior is as if the value of this field was 0b1.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

**Note**

Accesses to the Counter-timer Physical Secure timer registers are always enabled at EL3. These registers are not accessible at EL0.

This field resets to an architecturally UNKNOWN value.

**RW, bit [10]**

Execution state control for lower Exception levels.
Meaning

0b0  Lower levels are all AArch32.
0b1  The next lower level is AArch64.
    If EL2 is present:
      •  EL2 is AArch64.
      •  EL2 controls EL1 and EL0 behaviors.
    If EL2 is not present:
      •  EL1 is AArch64.
      •  EL0 is determined by the Execution state described in the current process state when executing at EL0.

If AArch32 state is not supported by the implementation at EL2 and AArch32 state is not supported by the implementation at EL1, then this bit is RAO/WI.

If AArch32 state is supported by the implementation at EL1, SCR_EL3.NS == 1 and AArch32 state is not supported by the implementation at EL2, the Effective value of this bit is 1.

If AArch32 state is supported by the implementation at EL1, ARMv8.4-SecEL2 is implemented and SCR_EL3.{EEL2, NS} == {1, 0}, the Effective value of this bit is 1.

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

SIF, bit [9]

When ARMv8.4-SecEL2 is implemented:

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from memory marked in the first stage of translation as being Non-secure. The possible values for this bit are:

<table>
<thead>
<tr>
<th>SIF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state instruction fetches from memory marked in the first stage of translation as being Non-secure are permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure state instruction fetches from memory marked in the first stage of translation as being Non-secure are not permitted.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory.

<table>
<thead>
<tr>
<th>SIF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state instruction fetches from Non-secure memory are permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure state instruction fetches from Non-secure memory are not permitted.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

This field resets to an architecturally UNKNOWN value.

HCE, bit [8]

Hypervisor Call instruction enable. Enables HVC instructions at EL3 and, if EL2 is enabled in the current Security state, at EL2 and EL1, in both Execution states, reported using EC syndrome value 0x00.

<table>
<thead>
<tr>
<th>HCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>HVC instructions are UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>HVC instructions are enabled at EL3, EL2, and EL1.</td>
</tr>
</tbody>
</table>

Note
HVC instructions are always *UNDEFINED* at EL0 and, if Secure EL2 is disabled, at Secure EL1. Any resulting exception is taken from the current Exception level to the current Exception level.

If EL2 is not implemented, this bit is *RES0*. This field resets to an architecturally *UNKNOWN* value.

**SMD, bit [7]**

Secure Monitor Call disable. Disables SMC instructions at EL1 and above, from both Security states and both Execution states, reported using EC syndrome value 0x00.

<table>
<thead>
<tr>
<th>SMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SMC instructions are enabled at EL3, EL2 and EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>SMC instructions are <em>UNDEFINED</em>.</td>
</tr>
</tbody>
</table>

**Note**

SMC instructions are always *UNDEFINED* at EL0. Any resulting exception is taken from the current Exception level to the current Exception level.

If **HCR_EL2** TSC or **HCR**.TSC traps attempted EL1 execution of SMC instructions to EL2, that trap has priority over this disable.

This field resets to an architecturally *UNKNOWN* value.

**Bit [6]**

Reserved, *RES0*.

**Bits [5:4]**

Reserved, *RES1*.

**EA, bit [3]**

External Abort and SError interrupt routing.

<table>
<thead>
<tr>
<th>EA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When executing at Exception levels below EL3, External aborts and SError interrupts are not taken to EL3. In addition, when executing at EL3:</td>
</tr>
<tr>
<td></td>
<td>• SError interrupts are not taken.</td>
</tr>
<tr>
<td></td>
<td>• External aborts are taken to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>When executing at any Exception level, External aborts and SError interrupts are taken to EL3.</td>
</tr>
</tbody>
</table>

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers' Model).

This field resets to an architecturally *UNKNOWN* value.

**FIQ, bit [2]**

Physical FIQ Routing.
**SCR_EL3**, Secure Configuration Register

<table>
<thead>
<tr>
<th>FIQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When executing at Exception levels below EL3, physical FIQ interrupts are not taken to EL3. When executing at EL3, physical FIQ interrupts are not taken.</td>
</tr>
<tr>
<td>0b1</td>
<td>When executing at any Exception level, physical FIQ interrupts are taken to EL3.</td>
</tr>
</tbody>
</table>

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

**IRQ, bit [1]**

Physical IRQ Routing.

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When executing at Exception levels below EL3, physical IRQ interrupts are not taken to EL3. When executing at EL3, physical IRQ interrupts are not taken.</td>
</tr>
<tr>
<td>0b1</td>
<td>When executing at any Exception level, physical IRQ interrupts are taken to EL3.</td>
</tr>
</tbody>
</table>

For more information, see 'Asynchronous exception routing' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1.

This field resets to an architecturally UNKNOWN value.

**NS, bit [0]**

Non-secure bit.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Indicates that EL0 and EL1 are in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Indicates that Exception levels lower than EL3 are in Non-secure state, and so memory accesses from those Exception levels cannot access Secure memory.</td>
</tr>
</tbody>
</table>

When SCR_EL3. {EEL2, NS} == {1, 0}, then EL2 is using AArch64 and in Secure state.

This field resets to an architecturally UNKNOWN value.

**Accessing the SCR_EL3**

Accesses to this register use the following encodings:

```
MRS <Xt>, SCR_EL3
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCR_EL3;```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR_EL3 = X[t];
**SCTLR_EL1, System Control Register (EL1)**

The SCTLR_EL1 characteristics are:

### Purpose

Provides top level control of the system, including its memory system, at EL1 and EL0.

### Configuration

AArch64 System register SCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SCTLR[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL1 using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

### Attributes

SCTLR_EL1 is a 64-bit register.

### Field descriptions

The SCTLR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>TME, bit [53]</td>
</tr>
<tr>
<td>52</td>
<td>TME0, bit [52]</td>
</tr>
</tbody>
</table>

**TME, bit [53]**

When TME is implemented:

Enables the Transactional Memory Extension at EL1.

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a TSTART instruction at EL1 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any TSTART instruction to be trapped.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TME0, bit [52]**

When TME is implemented:

Enables the Transactional Memory Extension at EL0.
Any attempt to execute a TSTART instruction at EL0 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0.

This control does not cause any TSTART instruction to be trapped.

If ARMv8.1-VHE is implemented, EL2 is implemented and enabled in the current Security state, and HCR_EL2.\{E2H, TGE\} == \{1,1\}, this field has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]

When TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL1.

<table>
<thead>
<tr>
<th>TMT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any TSTART instruction to fail.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the TSTART instruction is executed at EL1, the transaction fails with a TRIVIAL failure cause.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]

When TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL0.

<table>
<thead>
<tr>
<th>TMT0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any TSTART instruction to fail.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause.</td>
</tr>
</tbody>
</table>

If ARMv8.1-VHE is implemented, EL2 is implemented and enabled in the current Security state, and HCR_EL2.\{E2H, TGE\} == \{1,1\}, this field has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [49:46]

When ARMv8.6-TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL1.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by SCTLR_EL1.nTWE as \(2^{\text{TWEDEL} + 8}\) cycles.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RES0.

**TWEDEn, bit [45]**

**When ARMv8.6-TWED is implemented:**

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by SCTLR_EL1.nTWE.

<table>
<thead>
<tr>
<th>TWEDEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The delay for taking a WFE trap is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>The delay for taking a WFE trap is at least the number of cycles defined in SCTLR_EL1.TWEDEL.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**DSSBS, bit [44]**

**When ARMv8.0-SSBS is implemented:**

Default PSTATE.SSBS value on Exception Entry. The defined values are:

<table>
<thead>
<tr>
<th>DSSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.SSBS is set to 0 on an exception to EL1</td>
</tr>
<tr>
<td>0b1</td>
<td>PSTATE.SSBS is set to 1 on an exception to EL1</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL1, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:
Reserved, RES0.

**ATA, bit [43]**

**When ARMv8.5-MemTag is implemented:**

Allocation Tag Access in EL1. When SCR_EL3.ATA=1 and HCR_EL2.ATA=1, controls EL1 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

<table>
<thead>
<tr>
<th>ATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to Allocation Tags is prevented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to Allocation Tags is not prevented.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**ATA0, bit [42]**

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL0. When SCR_EL3.ATA=1, HCR_EL2.ATA=1, and HCR_EL2.{E2H,TGE} != {1,1}, controls EL0 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

<table>
<thead>
<tr>
<th>ATA0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to Allocation Tags is prevented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to Allocation Tags is not prevented.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

**Note**

Software may change this control bit on a context switch.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**TCF, bits [41:40]**

When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL1. Controls the effect of tag check fails due to Loads and Stores in EL1.

<table>
<thead>
<tr>
<th>TCF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Tag check fails have no effect on the PE.</td>
</tr>
<tr>
<td>0b01</td>
<td>Tag check fails causes a synchronous exception.</td>
</tr>
<tr>
<td>0b10</td>
<td>Tag check fails are asynchronously accumulated.</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.
TCF0, bits [39:38]
When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL0. When HCR_EL2.E2H,TGE != {1,1}, controls the effect of tag check fails due to Loads and Stores in EL0.

<table>
<thead>
<tr>
<th>TCF0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Tag check fails have no effect on the PE.</td>
</tr>
<tr>
<td>0b01</td>
<td>Tag check fails causes a synchronous exception.</td>
</tr>
<tr>
<td>0b10</td>
<td>Tag check fails are asynchronously accumulated.</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

Note
Software may change this control bit on a context switch.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

ITFSB, bit [37]
When ARMv8.5-MemTag is implemented:

When synchronous exceptions are not being generated by Tag Check fails which are generated for Loads and Stores in EL0 or EL1, controls the auto-synchronization of Tag Check fails into TFSRE0_EL1 and TFSR_EL1.

<table>
<thead>
<tr>
<th>ITFSB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tag check fails are not synchronized on entry to EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tag check fails are synchronized on entry to EL1.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

BT1, bit [36]
When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL1.

<table>
<thead>
<tr>
<th>BT1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When the PE is executing at EL1, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the PE is executing at EL1, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.
BT0, bit [35]

When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL0.

<table>
<thead>
<tr>
<th>BT0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
</tbody>
</table>

When HCR_EL2.E2H == 1 && HCR_EL2.TGE == 1, the value of the SCTLR_EL1.BT0 has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [34:32]

Reserved, RES0.

EnIA, bit [31]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnIA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.</td>
</tr>
</tbody>
</table>

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
EnIB

<table>
<thead>
<tr>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer authentication (using the APIBKey_EL1 key) of instruction</td>
<td></td>
</tr>
<tr>
<td>addresses is not enabled.</td>
<td></td>
</tr>
<tr>
<td>Pointer authentication (using the APIBKey_EL1 key) of instruction</td>
<td></td>
</tr>
<tr>
<td>addresses is enabled.</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**LSMAOE, bit [29]**

**When ARMv8.2-LSMAOC is implemented:**

Load Multiple and Store Multiple Atomicity and Ordering Enable.

<table>
<thead>
<tr>
<th>LSMAOE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.E2H, TGE is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES1.

**nTLSMD, bit [28]**

**When ARMv8.2-LSMAOC is implemented:**

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

<table>
<thead>
<tr>
<th>nTLSMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.</td>
</tr>
<tr>
<td>0b1</td>
<td>All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.E2H, TGE is {1,1}, this bit has no effect on execution at EL0.
In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES1**.

**EnDA, bit [27]**

**When ARMv8.3-PAuth is implemented:**

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**UCI, bit [26]**

Traps EL0 execution of cache maintenance instructions, to EL1, or to EL2 when it is implemented and enabled for the current Security state and **HCR_EL2.TGE** is 1, from AArch64 state only, reported using EC syndrome value 0x18.

This applies to **DC CVAU**, **DC CIVAC**, **DC CVAC**, **DC CVAP**, and **IC IVAU**.

If ARMv8.2-DCCVADP is implemented, this trap also applies to **DC CVADP**.

If ARMv8.5-MemTag is implemented, this trap also applies to **DC CIGVAC**, **DC CIGDVAC**, **DC CVAC**, **DC CGVAC**, **DC CGDVAC**, **DC CGVAP**, and **DC CGDVAP**.

If ARMv8.2-DCCVADP and ARMv8.5-MemTag are implemented, this trap also applies to **DC CGVADP** and **DC CGDVADP**.

<table>
<thead>
<tr>
<th>UCI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Execution of the specified instructions at EL0 using AArch64 is trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of **HCR_EL2.{E2H, TGE}** is {1, 1}, this bit has no effect on execution at EL0.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.
If the Point of Unification is before any level of data cache, it is IMPLEMENTATIONDEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATIONDEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**EE, bit [25]**

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

The possible values of this bit are:

<table>
<thead>
<tr>
<th>EE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&amp;0 translation regime are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&amp;0 translation regime are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to an IMPLEMENTATIONDEFINED value.

**E0E, bit [24]**

Endianness of data accesses at EL0.

The possible values of this bit are:

<table>
<thead>
<tr>
<th>E0E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Explicit data accesses at EL0 are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Explicit data accesses at EL0 are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation only supports Little-endian accesses at EL0 then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0 then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**SPAN, bit [23]**

When ARMv8.1-PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1.

<table>
<thead>
<tr>
<th>SPAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.PAN is set to 1 on taking an exception to EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The value of PSTATE.PAN is left unchanged on taking an exception to EL1.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.
In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES1**.

**EIS, bit [22]**

When ARMv8.5-CSEH is implemented:

Exception Entry is Context Synchronizing. The defined values are:

<table>
<thead>
<tr>
<th>EIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The taking of an exception to EL1 is not a context synchronizing event.</td>
</tr>
<tr>
<td>0b1</td>
<td>The taking of an exception to EL1 is a context synchronizing event.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of **HCR_EL2** \(\{E2H, TGE\}\) is \{1,1\}, this bit has no effect on execution at EL0.

If **SCTLR_EL1.EIS** is set to 0b0:

- Indirect writes to **ESR_EL1**, **FAR_EL1**, **SPSR_EL1**, **ELR_EL1** are synchronized on exception entry to EL1, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of **SCTLR EL1.EIS**:

- Changes to the PSTATE information on entry to EL1.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES1**.

**IESB, bit [21]**

When ARMv8.2-IESB is implemented:

Implicit Error Synchronization event enable. Possible values are:

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>An implicit error synchronization event is added:</td>
</tr>
<tr>
<td></td>
<td>• At each exception taken to EL1.</td>
</tr>
<tr>
<td></td>
<td>• Before the operational pseudocode of each <strong>ERET</strong> instruction executed at EL1.</td>
</tr>
</tbody>
</table>

When the PE is in Debug state, the effect of this field is **CONSTRAINED UNPREDICTABLE**, and its **Effective** value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each **DCPSX** instruction taken to EL1 and before each **DRPS** instruction executed at EL1, in addition to the other cases where it is added.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When ARMv8.0-CSV2 is implemented:

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64. The defined values are:

<table>
<thead>
<tr>
<th>TSCXT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The value of SCXTNUM_EL0 is treated as 0.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable in the EL1&amp;0 translation regime is forced to XN for accesses from software executing at EL1 or EL0.</td>
</tr>
</tbody>
</table>

This bit applies only when SCTLR_EL1.M bit is set.

The WXN bit is permitted to be cached in a TLB.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, from both Execution states, reported using EC syndrome value 0x01.

<table>
<thead>
<tr>
<th>nTWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a WFE instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note
Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of `HCR_EL2` {E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**Bit [17]**

Reserved, RES0.

**nTWI, bit [16]**

Traps EL0 execution of WFI instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and `HCR_EL2`.TGE is 1, from both Execution states, reported using EC syndrome value 0x01.

<table>
<thead>
<tr>
<th>nTWI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a WFI instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When ARMv8.1-VHE is implemented, and the value of `HCR_EL2` {E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**UCT, bit [15]**

Traps EL0 accesses to the `CTR_EL0` to EL1, or to EL2 when it is implemented and enabled for the current Security state and `HCR_EL2`.TGE is 1, from AArch64 state only, reported using EC value 0x18.

<table>
<thead>
<tr>
<th>UCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to the <code>CTR_EL0</code> from EL0 using AArch64 are trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of `HCR_EL2` {E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**DZE, bit [14]**

Traps EL0 execution of `DC ZVA` instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and `HCR_EL2`.TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.

If ARMv8.5-MemTag is implemented, this trap also applies to `DC GVA` and `DC GZVA`. 

---

SCTLR_EL1, System Control Register (EL1)
**DZE**

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped.</td>
</tr>
<tr>
<td><strong>0b1</strong> This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**EnDB, bit [13]**

*When ARMv8.3-PAuth is implemented:*

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td><strong>0b1</strong> Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL1, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**I, bit [12]**

Instruction access Cacheability control, for accesses at EL0 and EL1:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong> All instruction access to Normal memory from EL0 and EL1 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&amp;0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.</td>
</tr>
<tr>
<td><strong>0b1</strong> This control has no effect on the Cacheability of instruction access to Normal memory from EL0 and EL1. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&amp;0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.</td>
</tr>
</tbody>
</table>

When the value of the HCR_EL2 DC bit is 1, then instruction access to Normal memory from EL0 and EL1 are Cacheable regardless of the value of the SCTLR_EL1.I bit.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2 {E2H, TGE} is {1, 1}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to 0.
**EOS, bit [11]**

When ARMv8.5-CSEH is implemented:

Exception Exit is Context Synchronizing. The defined values are:

<table>
<thead>
<tr>
<th>EOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An exception return from EL1 is not a context synchronizing event</td>
</tr>
<tr>
<td>0b1</td>
<td>An exception return from EL1 is a context synchronizing event</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of \( \text{HCR} \_\text{EL2} \) \{E2H, TGE\} is \{1,1\}, this bit has no effect on execution at EL0.

If SCTLR\_EL1.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR\_EL1.EOS:

- The indirect write of the PSTATE and PC values from SPSR\_EL1 and ELR\_EL1 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES1.

**EnRCTX, bit [10]**

When ARMv8.0-PredInv is implemented:

Enable EL0 Access to the AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions, and the AArch64 CFP RCTX, DVP RCT and CPP RCTX instructions. The defined values are:

<table>
<thead>
<tr>
<th>EnRCTX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 access to these instructions is disabled, and these instructions are trapped to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 access to these instructions is enabled.</td>
</tr>
</tbody>
</table>

When ARMv8.1-VHE is implemented, and the value of \( \text{HCR} \_\text{EL2} \) \{E2H, TGE\} is \{1,1\}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**UMA, bit [9]**

User Mask Access. Traps EL0 execution of MSR and MRS instructions that access the PSTATE.{D, A, I, F} masks to EL1, or to EL2 when it is implemented and enabled for the current Security state and \( \text{HCR} \_\text{EL2} \).TGE is 1, from AArch64 state only, reported using EC syndrome value 0x18.
UMA, bit [7]

Any attempt at EL0 using AArch64 to execute an MRS, MSR(REGISTER), or MSR(IMMEDIATE) instruction that accesses the DAIF is trapped.

0b0 This control does not cause any instructions to be trapped.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

<table>
<thead>
<tr>
<th>SED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SETEND instruction execution is enabled at EL0 using AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>SETEND instructions are UNDEFINED at EL0 using AArch32 and any attempt at EL0 to access a SETEND instruction generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, reported using EC syndrome value 0x00.</td>
</tr>
</tbody>
</table>

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

If EL0 cannot use AArch32, this bit is RES1.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.
### ITD

**Meaning**

| 0b0 | All IT instruction functionality is enabled at EL0 using AArch32. |
| 0b1 | Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED and generates an exception, reported using EC syndrome value 0x0, to EL1 or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1: |
|     | - All encodings of the IT instruction with hw1[3:0]!=1000. |
|     | - All encodings of the subsequent instruction with the following values for hw1: |
|     |   - 0b11xxxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. |
|     |   - 0b1011xxxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5. |
|     |   - 0b10100xxxxxxxxxxx: ADD Rd, PC, #imm |
|     |   - 0b01001xxxxxxxxxxx: LDR Rd, [PC, #imm] |
|     |   - 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. |
|     |   - 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. |
|     | These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block. |
|     | It is IMPLEMENTATION DEFINED whether the IT instruction is treated as: |
|     |   - A 16-bit instruction, that can only be followed by another 16-bit instruction. |
|     |   - The first half of a 32-bit instruction. |
|     | This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction. |

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see ‘Changes to an ITD control by an instruction in an IT block’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.2.4

If EL0 cannot use AArch32, this bit is RES1.

**nAA, bit [6]**

*When ARMv8.4-LSE is implemented:*

Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.

| 0b0 | LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses. |
| 0b1 | This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes. |
When ARMv8.1-VHE is implemented, and the value of `HCR_EL2.{E2H, TGE}` is \{1, 1\}, this bit has no effect on execution at \( \text{EL0} \).

In a system where the PE resets into EL1, this field resets to an architecturally `UNKNOWN` value.

**Otherwise:**

Reserved, \( \text{RES0} \).

**CP15BEN, bit [5]**

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from \( \text{EL0} \):

<table>
<thead>
<tr>
<th>CP15BEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>( \text{EL0 using AArch32}: \text{EL0 execution of the CP15DMB, CP15DSB, and CP15ISB} ) instructions is \text{UNDEFINED} and generates an exception to ( \text{EL1} ), or to ( \text{EL2} ) when it is implemented and enabled for the current Security state and ( \text{HCR_EL2.TGE} ) is 1. The exception is reported using EC syndrome value 0x00.</td>
</tr>
<tr>
<td>0b1</td>
<td>( \text{EL0 using AArch32}: \text{EL0 execution of the CP15DMB, CP15DSB, and CP15ISB} ) instructions is enabled.</td>
</tr>
</tbody>
</table>

If \( \text{EL0} \) cannot use AArch32, this bit is \( \text{RES0} \).

CP15BEN is optional, but if it is implemented in the `SCTLR` then it must also be implemented in the `SCTLR_EL1`. If it is not implemented then this bit is RAO/WI.

When ARMv8.1-VHE is implemented, and the value of `HCR_EL2.{E2H, TGE}` is \{1, 1\}, this bit has no effect on execution at \( \text{EL0} \).

In a system where the PE resets into EL1, this field resets to an architecturally `UNKNOWN` value.

**SA0, bit [4]**

SP Alignment check enable for \( \text{EL0} \). When set to 1, if a load or store instruction executed at \( \text{EL0} \) uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see ‘SP alignment checking’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers’ Model).

When ARMv8.1-VHE is implemented, and the value of `HCR_EL2.{E2H, TGE}` is \{1, 1\}, this bit has no effect on execution at \( \text{EL0} \).

In a system where the PE resets into EL1, this field resets to an architecturally `UNKNOWN` value.

**SA, bit [3]**

SP Alignment check enable. When set to 1, if a load or store instruction executed at \( \text{EL1} \) uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see ‘SP alignment checking’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers’ Model).

When ARMv8.1-VHE is implemented, and the value of `HCR_EL2.{E2H, TGE}` is \{1, 1\}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to an architecturally `UNKNOWN` value.

**C, bit [2]**

Cacheability control, for data accesses.
C, bit [0]

All data access to Normal memory from EL0 and EL1, and all Normal memory accesses to the EL1&0 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache.

0b0
This control has no effect on the Cacheability of:
• Data access to Normal memory from EL0 and EL1.
• Normal memory accesses to the EL1&0 stage 1 translation tables.

0b1
When the value of the \texttt{HCR\_EL2}.DC bit is 1, the PE ignores \texttt{SCLTR.C}. This means that Non-secure EL0 and Non-secure EL1 data accesses to Normal memory are Cacheable.

When ARMv8.1-VHE is implemented, and the value of \texttt{HCR\_EL2}.{E2H, TGE} is \{1, 1\}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL1 and EL0.

0b0
Alignment fault checking disabled when executing at EL1 or EL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.

0b1
Alignment fault checking enabled when executing at EL1 or EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

When ARMv8.1-VHE is implemented, and the value of \texttt{HCR\_EL2}.{E2H, TGE} is \{1, 1\}, this bit has no effect on execution at EL0.

In a system where the PE resets into EL1, this field resets to an architecturally \texttt{UNKNOWN} value.

M, bit [0]

MMU enable for EL1 and EL0 stage 1 address translation. Possible values of this bit are:

0b0
EL1 and EL0 stage 1 address translation disabled. See the \texttt{SCTLR\_EL1.I} field for the behavior of instruction accesses to Normal memory.

0b1
EL1 and EL0 stage 1 address translation enabled.

If the value of \texttt{HCR\_EL2}.{DC, TGE} is not \{0, 0\} then in Non-secure state the PE behaves as if the value of the \texttt{SCTLR\_EL1.M} field is 0 for all purposes other than returning the value of a direct read of the field.

When ARMv8.1-VHE is implemented, and the value of \texttt{HCR\_EL2}.{E2H, TGE} is \{1, 1\}, this bit has no effect on the PE.

In a system where the PE resets into EL1, this field resets to 0.

Accessing the \texttt{SCTLR\_EL1}

When \texttt{HCR\_EL2}.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic \texttt{SCTLR\_EL1} or \texttt{SCTLR\_EL12} are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, SCTLR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRT_EL2.SCTLR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x110];
  else
    return SCTLR_EL1;
  elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
      return SCTLR_EL2;
    else
      return SCTLR_EL1;
    elsif PSTATE.EL == EL3 then
      return SCTLR_EL1;

  MSR SCTLR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.SCTLR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x110] = X[t];
  else
    SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    SCTLR_EL2 = X[t];
  else
    SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  SCTLR_EL1 = X[t];

MRS <Xt>, SCTLR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x110];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return SCTLR_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return SCTLR_EL1;
  else
    UNDEFINED;

MSR SCTLR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    NVMem[0x110] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    SCTLR_EL1 = X[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    SCTLR_EL1 = X[t];
  else
    UNDEFINED;
SCTLR_EL2, System Control Register (EL2)

The SCTLR_EL2 characteristics are:

**Purpose**

Provides top level control of the system, including its memory system, at EL2.

When ARMv8.1-VHE is implemented, and the value of HCR_EL2.E2H, TGE is {1, 1}, these controls apply also to execution at EL0.

**Configuration**

AArch64 System register SCTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HSCTLR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SCTLR_EL2 is a 64-bit register.

**Field descriptions**

The SCTLR_EL2 bit assignments are:

**When HCR_EL2.E2H != 0b1 or HCR_EL2.TGE != 0b1:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | TME | RES0 | TMT | RES0 | DSSBS | ATA | RES0 | TCF | RES0 | ITF5 | EnIA | EnIB | RES1 | EnDA | RES0 | EE | RES0 | RES1 | EIS | IESB | RES0 | WXN | RES1 | RES0 | RES0 | EnDB | EOS | RES0 | nAA | RES1 | RES0 | RES0 | RES1 | RES0 | RES0 | RES0 | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 |

This format applies in all Armv8.0 implementations, and from Armv8.1 when the Effective value of HCR_EL2.E2H, TGE} != {1, 1}.

**Bits [63:54]**

Reserved, RES0.

**TME, bit [53]**

When TME is implemented:

Enables the Transactional Memory Extension at EL2.

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a TSTART instruction at EL2 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any TSTART instruction to be trapped.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**Bit [52]**

Reserved, RES0.

**TMT, bit [51]**

When TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL2.

<table>
<thead>
<tr>
<th>TMT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any TSTART instruction to fail.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the TSTART instruction is executed at EL2, the transaction fails with a TRIVIAL failure cause.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [50:45]**

Reserved, RES0.

**DSSBS, bit [44]**

When ARMv8.0-SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

<table>
<thead>
<tr>
<th>DSSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.SSBS is set to 0 on an exception to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>PSTATE.SSBS is set to 1 on an exception to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

**ATA, bit [43]**

When ARMv8.5-MemTag is implemented:

Allocation Tag Access. When SCR_EL3.ATA=1, controls EL2 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.
<table>
<thead>
<tr>
<th>ATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to Allocation Tags is prevented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to Allocation Tags is not prevented.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TCF, bits [41:40]

When ARMv8.5-MemTag is implemented:

Tack Check Fail in EL2. Controls the effect of tag check fails due to Loads and Stores in EL2.

<table>
<thead>
<tr>
<th>TCF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Tag check fails have no effect on the PE.</td>
</tr>
<tr>
<td>0b01</td>
<td>Tag check fails causes a synchronous exception.</td>
</tr>
<tr>
<td>0b10</td>
<td>Tag check fails are asynchronously accumulated.</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

ITFSB, bit [37]

When ARMv8.5-MemTag is implemented:

When synchronous exceptions are not being generated by Tag Check fails which are generated for Loads and Stores in EL0, EL1 or EL2, controls the auto-synchronisation of Tag Check fails into TFSRE0_EL1, TFSR_EL1 and TFSR_EL2.

<table>
<thead>
<tr>
<th>ITFSB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tag check fails are not synchronized on entry to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tag check fails are synchronized on entry to EL2.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

BT, bit [36]

When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL2.
BT

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### Bits [35:32]

Reserved, RES0.

### EnIA, bit [31]

**When ARMv8.3-PAuth is implemented:**

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnIA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### EnIB, bit [30]

**When ARMv8.3-PAuth is implemented:**

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**
This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:28]

Reserved, RES1.

EnDA, bit [27]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

EE, bit [25]

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime.

The possible values of this bit are:
**EE**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&amp;0 translation regime, and stage 2 translation table walks in the EL1&amp;0 translation regime are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&amp;0 translation regime, and stage 2 translation table walks in the EL1&amp;0 translation regime are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is `RES0`.

If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is `RES1`.

The EE bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an **IMPLEMENTATION DEFINED** value.

**Bit [24]**

Reserved, `RES0`.

**Bit [23]**

Reserved, `RES1`.

**EIS, bit [22]**

### When ARMv8.5-CSEH is implemented:

Exception entry is a context synchronization event. The defined values are:

<table>
<thead>
<tr>
<th>EIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The taking of an exception to EL2 is not a context synchronization event.</td>
</tr>
<tr>
<td>0b1</td>
<td>The taking of an exception to EL2 is a context synchronization event.</td>
</tr>
</tbody>
</table>

If `SCTLR_EL2.EIS` is set to 0b0:

- Indirect writes to `ESR_EL2, FAR_EL2, SPSR_EL2, ELR_EL2, HPFAR_EL2` are synchronized on exception entry to EL2, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of `SCTLR_EL2.EIS`:

- Changes to the PSTATE information on entry to EL2.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, `RES1`.

**IESB, bit [21]**

### When ARMv8.2-IESB is implemented:

Implicit Error Synchronization event enable.
When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL2 and before each DRPS instruction executed at EL2, in addition to the other cases where it is added.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bit [20]**

Reserved, RES0.

**WXN, bit [19]**

Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can force all memory regions that are writable to be treated as XN:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable in the EL2 or EL2&amp;0 translation regime is forced to XN for accesses from software executing at EL2.</td>
</tr>
</tbody>
</table>

This bit applies only when SCTLR_EL2.M bit is set.

The WXN bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**Bit [18]**

Reserved, RES1.

**Bit [17]**

Reserved, RES0.

**Bit [16]**

Reserved, RES1.

**Bits [15:14]**

Reserved, RES0.

**EnDB, bit [13]**

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL2&0 translation regime.
For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnDB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**I, bit [12]**

Instruction access Cacheability control, for accesses at EL2:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All instruction access to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&amp;0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control has no effect on the Cacheability of instruction access to Normal memory from EL2. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&amp;0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.</td>
</tr>
</tbody>
</table>

This bit has no effect on the EL1&0 or EL3 translation regimes.

In a system where the PE resets into EL2, this field resets to 0.

**EOS, bit [11]**

*When ARMv8.5-CSEH is implemented:*

Exception exit is a context synchronization Event.

<table>
<thead>
<tr>
<th>EOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An exception return from EL2 is not a context synchronization event.</td>
</tr>
<tr>
<td>0b1</td>
<td>An exception return from EL2 is a context synchronization event.</td>
</tr>
</tbody>
</table>

If SCTLR_EL2.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL2 and ELR_EL2 on exception return is synchronized.
Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.

Exit from Debug state.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES1.

**Bits [10:7]**

Reserved, RES0.

**nAA, bit [6]**

When ARMv8.4-LSE is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL2 under certain conditions.

<table>
<thead>
<tr>
<th>nAA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLURH, STLR, STLURH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLURH, STLR, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Bits [5:4]**

Reserved, RES1.

**SA, bit [3]**

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL2 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers’ Model).

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**C, bit [2]**

Cacheability control, for data accesses.

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All data access to Normal memory from EL2, and all Normal memory accesses to the EL2 translation tables, are Non-cacheable for all levels of data and unified cache.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control has no effect on the Cacheability of:</td>
</tr>
<tr>
<td></td>
<td>• Data access to Normal memory from EL2.</td>
</tr>
<tr>
<td></td>
<td>• Normal memory accesses to the EL2 translation tables.</td>
</tr>
</tbody>
</table>
This bit has no effect on the EL1&0 or EL3 translation regimes.

In a system where the PE resets into EL2, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL2:

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Alignment fault checking disabled when executing at EL2. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Alignment fault checking enabled when executing at EL2. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.</td>
</tr>
</tbody>
</table>

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 stage 1 address translation. Possible values of this bit are:

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL2 stage 1 address translation disabled. See the SCTLR_EL2.I field for the behavior of instruction accesses to Normal memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 stage 1 address translation enabled.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

When HCR_EL2.E2H == 0b1 and HCR_EL2.TGE == 0b1:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:54</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

TME, bit [53]

When TME is implemented:

Enables the Transactional Memory Extension at EL2.

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a TSTART instruction at EL2 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any TSTART instruction to be trapped.</td>
</tr>
</tbody>
</table>
In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TME0, bit [52]**

**When TME is implemented:**

Enables the Transactional Memory Extension at EL0.

<table>
<thead>
<tr>
<th>TME0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a TSTART instruction at EL0 is trapped to EL2, unless <strong>HCR_EL2.TME</strong> or <strong>SCR_EL3.TME</strong> causes TSTART instructions to be <strong>UNDEFINED</strong> at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any TSTART instruction to be trapped.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TMT, bit [51]**

**When TME is implemented:**

Forces a trivial implementation of the Transactional Memory Extension at EL2.

<table>
<thead>
<tr>
<th>TMT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any TSTART instruction to fail.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the TSTART instruction is executed at EL2, the transaction fails with a <strong>TRIVIAL</strong> failure cause.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TMT0, bit [50]**

**When TME is implemented:**

Forces a trivial implementation of the Transactional Memory Extension at EL0.

<table>
<thead>
<tr>
<th>TMT0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any TSTART instruction to fail.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the TSTART instruction is executed at EL0, the transaction fails with a <strong>TRIVIAL</strong> failure cause.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TWEDEL, bits [49:46]**
When ARMv8.6-TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by SCTLR_EL2.nTWE as $2^{(TWEDEL + 8)}$ cycles.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**TWEDEn, bit [45]**

When ARMv8.6-TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by SCTLR_EL2.nTWE.

<table>
<thead>
<tr>
<th>TWDEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The delay for taking a WFE trap is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>The delay for taking a WFE trap is at least the number of cycles defined in SCTLR_EL2.TWEDEL.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**DSSBS, bit [44]**

When ARMv8.0-SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

<table>
<thead>
<tr>
<th>DSSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.SSBS is set to 0 on an exception to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>PSTATE.SSBS is set to 1 on an exception to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

**ATA, bit [43]**

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL2. When SCR_EL3.ATA=1, controls EL2 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.
ATA

<table>
<thead>
<tr>
<th>Meaning</th>
<th>ATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to Allocation Tags is prevented.</td>
<td>0b0</td>
</tr>
<tr>
<td>Access to Allocation Tags is not prevented.</td>
<td>0b1</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**ATA0, bit [42]**

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL0. When `SCR_EL3.ATA=1`, controls EL0 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>ATA0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access to Allocation Tags is prevented.</td>
<td>0b0</td>
</tr>
<tr>
<td>Access to Allocation Tags is not prevented.</td>
<td>0b1</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

**Note**

Software may change this control bit on a context switch.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TCF, bits [41:40]**

When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL2. Controls the effect of tag check fails due to Loads and Stores in EL2.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>TCF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tag check fails have no effect on the PE.</td>
<td>0b00</td>
</tr>
<tr>
<td>Tag check fails causes a synchronous exception.</td>
<td>0b01</td>
</tr>
<tr>
<td>Tag check fails are asynchronously accumulated.</td>
<td>0b10</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**TCF0, bits [39:38]**

When ARMv8.5-MemTag is implemented:

Tag Check Fail in EL0. Controls the effect of tag check fails due to Loads and Stores in EL0.

<table>
<thead>
<tr>
<th>TCF0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Tag check fails have no effect on the PE.</td>
</tr>
<tr>
<td>0b01</td>
<td>Tag check fails causes a synchronous exception.</td>
</tr>
<tr>
<td>0b10</td>
<td>Tag check fails are asynchronously accumulated.</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

**Note**

Software may change this control bit on a context switch.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**ITFSB, bit [37]**

When ARMv8.5-MemTag is implemented:

When synchronous exceptions are not being generated by Tag Check fails which are generated for Loads and Stores in EL0, EL1 or EL2, controls the auto-synchronization of Tag Check fails into TFSRE0_EL1, TFSR_EL1 and TFSR_EL2.

<table>
<thead>
<tr>
<th>ITFSB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tag check fails are not synchronized on entry to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tag check fails are synchronized on entry to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**BT1, bit [36]**

When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL2.

<table>
<thead>
<tr>
<th>BT1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When the PE is executing at EL2, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the PE is executing at EL2, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
When ARMv8.5-BTI is implemented:

PAC Branch Type compatibility at EL0.

<table>
<thead>
<tr>
<th>BT0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

Bits [34:32]

Reserved, RES0.

EnIA, bit [31]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnIA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.</td>
</tr>
</tbody>
</table>

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
Meaning

0b0  Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.
0b1  Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When ARMv8.2-LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

Meaning

0b0  For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.
0b1  The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0.

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]

When ARMv8.2-LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

Meaning

0b0  All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.
0b1  All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES1.

**EnDA, bit [27]**

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**UCI, bit [26]**

Traps EL0 execution of cache maintenance instructions to EL2, from AArch64 state only. This applies to DC CVAU, DC CIVAC, DC CVAC, DC CVAP, and IC IVAU.

If ARMv8.2-DCCVADP is implemented, this trap also applies to DC CVADP.

If ARMv8.5-MemTag is implemented, this trap also applies to DC CIGVAC, DC CIGDVAC, DC CGVAC, DC CGDVAC, DC CGVAP, and DC CGDVAP.

If ARMv8.2-DCCVADP and ARMv8.5-MemTag are implemented, this trap also applies to DC CGVADP and DC CGDVADP.

<table>
<thead>
<tr>
<th>UCI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.
**EE, bit [25]**

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL2&0 translation regime.

The possible values of this bit are:

<table>
<thead>
<tr>
<th>EE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&amp;0 translation regime, and stage 2 translation table walks in the EL2&amp;0 translation regime are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&amp;0 translation regime, and stage 2 translation table walks in the EL2&amp;0 translation regime are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an IMPLEMENTATION DEFINED value.

**E0E, bit [24]**

Endianness of data accesses at EL0.

<table>
<thead>
<tr>
<th>E0E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Explicit data accesses at EL0 are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Explicit data accesses at EL0 are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation only supports Little-endian accesses at EL0 then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0 then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**SPAN, bit [23]**

Set Privileged Access Never, on taking an exception to EL2.

<table>
<thead>
<tr>
<th>SPAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.PAN is set to 1 on taking an exception to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>The value of PSTATE.PAN is left unchanged on taking an exception to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**EIS, bit [22]**

*When ARMv8.5-CSEH is implemented:*

Exception Entry is a context synchronization event.

<table>
<thead>
<tr>
<th>EIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The taking of an exception to EL2 is not a context synchronization event.</td>
</tr>
<tr>
<td>0b1</td>
<td>The taking of an exception to EL2 is a context synchronization event.</td>
</tr>
</tbody>
</table>

If SCTLR_EL2.EIS is set to 0b0:
Indirect writes to ESR_EL2, FAR_EL2, SPSR_EL2, ELR_EL2, HPFAR_EL2 are synchronized on exception entry to EL2, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.

Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.

Exception Catch debug events are synchronous debug events.

DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EIS:

- Changes to the PSTATE information on entry to EL2.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES1.

**IESB, bit [21]**

*When ARMv8.2-IESB is implemented:*

Implicit Error Synchronization event Enable.

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>An implicit error synchronization event is added:</td>
</tr>
<tr>
<td></td>
<td>- After each exception taken to EL2.</td>
</tr>
<tr>
<td></td>
<td>- Before the operational pseudocode of each ERET instruction executed at EL2.</td>
</tr>
</tbody>
</table>

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSx instruction taken to EL2 and before each DRPS instruction executed at EL2, in addition to the other cases where it is added.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**TSCXT, bit [20]**

*When ARMv8.0-CSV2 is implemented:*

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

<table>
<thead>
<tr>
<th>TSCXT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 access to SCXTNUM_EL0 is not disabled by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the SCXTNUM_EL0 value is treated at 0.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES1.
WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable in the EL2 or EL2&amp;0 translation regime is forced to XN for accesses from software executing at EL2.</td>
</tr>
</tbody>
</table>

This bit applies only when SCTLR_EL2.M bit is set.

The WXN bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to EL2, from both Execution states.

<table>
<thead>
<tr>
<th>nTWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a WFE instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to EL2, from both Execution states.

<table>
<thead>
<tr>
<th>nTWI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a WFI instruction at EL0 is trapped EL2, if the instruction would otherwise have caused the PE to enter a low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.
In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**UCT, bit [15]**

Traps EL0 accesses to the CTR_EL0 to EL2, from AArch64 state only.

<table>
<thead>
<tr>
<th>UCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to the CTR_EL0 from EL0 using AArch64 are trapped to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**DZE, bit [14]**

Traps EL0 execution of DC ZVA instructions to EL2, from AArch64 state only.

If ARMv8.5-MemTag is implemented, this trap also applies to DC GVA and DC GZVA.

<table>
<thead>
<tr>
<th>DZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions that this trap applies to are not supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**EnDB, bit [13]**

*When ARMv8.3-PAuth is implemented:*

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL2&0 translation regime.

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>EnDB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**I, bit [12]**

Instruction access Cacheability control, for accesses at EL2 and EL0:

---

Page 1371
### SCTLR_EL2, System Control Register (EL2)

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All instruction access to Normal memory from EL2 and EL0 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2&amp;0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control has no effect on the Cacheability of instruction access to Normal memory from EL2 and EL0. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2&amp;0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory. This bit has no effect on the EL3 translation regimes.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

### EOS, bit [11]

**When ARMv8.5-CSEH is implemented:**

Exception exit is a context synchronization event.

<table>
<thead>
<tr>
<th>EOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An exception return from EL2 is not a context synchronization event.</td>
</tr>
<tr>
<td>0b1</td>
<td>An exception return from EL2 is a context synchronization event.</td>
</tr>
</tbody>
</table>

If SCTLR_EL2.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL2 and ELR_EL2 on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES1.

### EnRCTX, bit [10]

**When ARMv8.0-PredInv is implemented:**

Enable EL0 Access to the AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions, and the AArch64 CFP RCTX, DVP RCT and CPP RCTX System instructions.

<table>
<thead>
<tr>
<th>EnRCTX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 access to these instructions is disabled, and these instructions are trapped to EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 access to these instructions is enabled.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES0.
Bit [9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

<table>
<thead>
<tr>
<th>SED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SETEND instruction execution is enabled at EL0 using AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>SETEND instructions are UNDEFINED at EL0 using AArch32.</td>
</tr>
</tbody>
</table>

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

If EL0 cannot use AArch32, this bit is RES1.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

<table>
<thead>
<tr>
<th>ITD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All IT instruction functionality is enabled at EL0 using AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED:</td>
</tr>
</tbody>
</table>

- All encodings of the IT instruction with hw1[3:0] != 1000.
- All encodings of the subsequent instruction with the following values for hw1:
  - 0b11xxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM.
  - 0b1011xxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5.
  - 0b10100xxxxxxxx: ADD Rd, PC, #imm
  - 0b01001xxxxxxx: LDR Rd, [PC, #imm]
  - 0b01001xxxxxxx: ADD Rd, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC.
  - 0b010001xxx1111xxx: ADD PC, Rn; CMP PC, Rm; MOV PC, Rm. This pattern also covers UNPREDICTABLE cases with BLX Rn.

These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.

It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:
- A 16-bit instruction, that can only be followed by another 16-bit instruction.
- The first half of a 32-bit instruction.

This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.

An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONstrained UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.2.4

If EL0 cannot use AArch32, this bit is RES1.

ITD is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1. If it is not implemented then this bit is RAZ/WI.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.
When ARMv8.4-LSE is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL2 and EL0 under certain conditions.

<table>
<thead>
<tr>
<th>nAA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLUR, STLURH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLUR, STLURH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**CP15BEN, bit [5]**

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

<table>
<thead>
<tr>
<th>CP15BEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is <strong>UNDEFINED</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 using AArch32: EL0 execution of the CP15DMB, CP15DSB, and CP15ISB instructions is enabled.</td>
</tr>
</tbody>
</table>

If EL0 cannot use AArch32, this bit is RES0.

CP15BEN is optional, but if it is implemented in the SCTLR then it must also be implemented in the SCTLR_EL1. If it is not implemented then this bit is RAO/WI.

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**SA0, bit [4]**

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers’ Model).

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**SA, bit [3]**

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL2 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers’ Model).

In a system where the PE resets into EL2, this field resets to an architecturally **UNKNOWN** value.

**C, bit [2]**

Cacheability control, for data accesses.
Meaning
0b0 All data access to Normal memory from EL2 and EL0, and all Normal memory accesses to the EL2&0 translation tables, are Non-cacheable for all levels of data and unified cache.
0b1 This control has no effect on the Cacheability of:
- Data access to Normal memory from EL2 and EL0.
- Normal memory accesses to the EL2&0 translation tables.

This bit has no effect on the EL3 translation regimes.

In a system where the PE resets into EL2, this field resets to 0.

Alignment check enable. This is the enable bit for Alignment fault checking at EL2 and EL0.

Meaning
0b0 Alignment fault checking disabled when executing at EL2 and EL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.
0b1 Alignment fault checking enabled when executing at EL2 and EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

MMU enable for EL2&0 stage 1 address translation. Possible values of this bit are:

Meaning
0b0 EL2&0 stage 1 address translation disabled. See the SCTLR_EL2.I field for the behavior of instruction accesses to Normal memory.
0b1 EL2&0 stage 1 address translation enabled.

In a system where the PE resets into EL2, this field resets to 0.

Accessing the SCTLR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic SCTLR_EL2 or SCTLR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

\[
\text{MRS} \ <Xt>, \ \text{SCTLR\_EL2}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SCTLR_EL2;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL2;

MSR SCTLR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SCTLR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SCTLR_EL2 = X[t];

MRS <Xt>, SCTLR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.SCTLR_EL1 == '1' then
        return NVMem[0x110];
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SCTLR_EL2;
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL1;

MSR SCTLR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.SCTLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x110] = X[t];
    else
        SCTLR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SCTLR_EL2 = X[t];
    else
        SCTLR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t];
SCTLR_EL3, System Control Register (EL3)

The SCTLR_EL3 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL3.

Configuration

This register is present only when HaveEL(EL3). Otherwise, direct accesses to SCTLR_EL3 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL3 using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SCTLR_EL3 is a 64-bit register.

Field descriptions

The SCTLR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>TME</td>
<td>RES0</td>
<td>TMT</td>
<td>RES0</td>
<td>DSSBS</td>
<td>ATA</td>
<td>RES0</td>
<td>TCF</td>
<td>RES0</td>
<td>TIF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EnIA</td>
<td>EnIB</td>
<td>RES1</td>
<td>EnDA</td>
<td>RES0</td>
<td>EE</td>
<td>RES0</td>
<td>RES1</td>
<td>EIS</td>
<td>RES0</td>
<td>WXN</td>
<td>RES1</td>
<td>RES0</td>
<td>RES0</td>
<td>EnDB</td>
<td></td>
<td>EOS</td>
<td>RES0</td>
<td>RES0</td>
<td>hAA</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:54]

Reserved, RES0.

TME, bit [53]

When TME is implemented:

Enables the Transactional Memory Extension at EL3.

<table>
<thead>
<tr>
<th>TME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a TSTART instruction at EL3 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any TSTART instruction to be trapped.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [52]

Reserved, RES0.

TMT, bit [51]
When TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL3.

<table>
<thead>
<tr>
<th>TMT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any TSTART instruction to fail.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the TSTART instruction is executed at EL3, the transaction</td>
</tr>
<tr>
<td></td>
<td>fails with a TRIVIAL failure cause.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [50:45]

Reserved, RES0.

DSSBS, bit [44]

When ARMv8.0-SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

<table>
<thead>
<tr>
<th>DSSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.SSBS is set to 0 on an exception to EL3</td>
</tr>
<tr>
<td>0b1</td>
<td>PSTATE.SSBS is set to 1 on an exception to EL3</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When ARMv8.5-MemTag is implemented:

Allocation Tag Access in EL3. Controls EL3 access to Allocation Tags.

When access to Allocation Tags is prevented:

- Instructions which Load or Store data are Unchecked.
- Instructions which Load or Store Allocation Tags treat the Allocation Tag as RAZ/WI.
- Instructions which insert Logical Address Tags into addresses treat the Allocation Tag used to generate the Logical Address Tag as 0.
- Cache maintenance instructions which invalidate Allocation Tags from caches behave as the equivalent Clean and Invalidate operation on Allocation Tags.

<table>
<thead>
<tr>
<th>ATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to Allocation Tags is prevented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to Allocation Tags is not prevented.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**Bit [42]**

Reserved, RES0.

**TCF, bits [41:40]**

*When ARMv8.5-MemTag is implemented:*

Tag Check Fail in EL3. Controls the effect of tag check fails due to Loads and Stores in EL3.

<table>
<thead>
<tr>
<th>TCF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tag check fails have no effect on the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tag check fails causes a synchronous exception.</td>
</tr>
<tr>
<td>0b10</td>
<td>Tag check fails are asynchronously accumulated.</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

In a system where the PE resets into EL3, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**Bits [39:38]**

Reserved, RES0.

**ITFSB, bit [37]**

*When ARMv8.5-MemTag is implemented:*

When asynchronous exceptions are being generated by Tag Check fails which are generated for Loads and Stores at any exception level, controls the auto-synchronisation of Tag Check fails into **TFSRE0_EL1** and **TFSR_ELx**.

<table>
<thead>
<tr>
<th>ITFSB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tag check fails are not synchronized on entry to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tag check fails are synchronized on entry to EL3.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**BT, bit [36]**

*When ARMv8.5-BTI is implemented:*

PAC Branch Type compatibility at EL3.

<table>
<thead>
<tr>
<th>BT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When the PE is executing at EL3, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the PE is executing at EL3, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**Bits [35:32]**

Reserved, RES0.

**EnIA, bit [31]**

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL3 translation regime.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>EnIA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled.</td>
</tr>
</tbody>
</table>

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**EnIB, bit [30]**

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL3 translation regime.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>EnIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled.</td>
</tr>
</tbody>
</table>

For more information, see ‘System register control of pointer authentication’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a
pointer to which a pointer authentication code has been added, and AuthIB
returns an authenticated copy of a pointer. When the field is 0, both of these
functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:28]

Reserved, RES1.

EnDA, bit [27]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL3 translation regime.

<table>
<thead>
<tr>
<th>EnDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

EE, bit [25]

Endianness of data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime.

<table>
<thead>
<tr>
<th>EE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is RES0.
If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an IMPLEMENTATION DEFINED value.

**Bit [24]**

Reserved, RES0.

**Bit [23]**

Reserved, RES1.

**EIS, bit [22]**

*When ARMv8.5-CSEH is implemented:*

Exception Entry is Context Synchronizing.

<table>
<thead>
<tr>
<th>EIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The taking of an exception to EL3 is not a context synchronizing event.</td>
</tr>
<tr>
<td>0b1</td>
<td>The taking of an exception to EL3 is a context synchronizing event.</td>
</tr>
</tbody>
</table>

If SCTLR_EL3.EIS is set to 0b0:

- Indirect writes to **ESR_EL3, FAR_EL3, SPSR_EL3, ELR_EL3** are synchronized on exception entry to EL3, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL3.EIS:

- Changes to the PSTATE information on entry to EL3.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Debug state exit.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

*Otherwise:*

Reserved, RES1.

**IESB, bit [21]**

*When ARMv8.2-IESB is implemented:*

Implicit Error Synchronization event enable.

<table>
<thead>
<tr>
<th>IESB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>An implicit error synchronization event is added:</td>
</tr>
<tr>
<td></td>
<td>• At each exception taken to EL3.</td>
</tr>
<tr>
<td></td>
<td>• Before the operational pseudocode of each ERET instruction executed at EL3.</td>
</tr>
</tbody>
</table>

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL3 and before each DRPS instruction executed at EL3, in addition to the other cases where it is added.
When ARMv8.4-DFE is implemented, and the Effective value of SCR_EL3.NMEA is 1, this field is ignored and its Effective value is 1.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL3 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable in the EL3 translation regime is forced to XN for accesses from software executing at EL3.</td>
</tr>
</tbody>
</table>

This bit applies only when SCTLRE_L3.M bit is set.

The WXN bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:14]

Reserved, RES0.

EnDB, bit [13]

When ARMv8.3-PAuth is implemented:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL3 translation regime.

<table>
<thead>
<tr>
<th>EnDB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled.</td>
</tr>
</tbody>
</table>

For more information, see 'System register control of pointer authentication' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Note
This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL3:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All instruction access to Normal memory from EL3 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control has no effect on the Cacheability of instruction access to Normal memory from EL3. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.</td>
</tr>
</tbody>
</table>

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

In a system where the PE resets into EL3, this field resets to 0.

EOS, bit [11]

When ARMv8.5-CSEH is implemented:

Exception Exit is Context Synchronizing.

<table>
<thead>
<tr>
<th>EOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An exception return from EL3 is not a context synchronizing event</td>
</tr>
<tr>
<td>0b1</td>
<td>An exception return from EL3 is a context synchronizing event</td>
</tr>
</tbody>
</table>

If SCTLR_EL3.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL3.EOS:

- The indirect write of the PSTATE and PC values from SPSR_EL3 and ELR_EL3 on exception return is synchronized.
- If the PE enters Debug state before the first instruction after an Exception return from EL3 to Non-secure state, any pending Halting debug event completes execution.
- The GIC behavior that allocates interrupts to FIQ or IRQ changes simultaneously with leaving the EL3 Exception level.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES1.

**Bits [10:7]**

Reserved, RES0.

**nAA, bit [6]**

When ARMv8.4-LSE is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL3 under certain conditions.

<table>
<thead>
<tr>
<th>nAA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [5:4]**

Reserved, RES1.

**SA, bit [3]**

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL3 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see ‘SP alignment checking’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D1 (The AArch64 System Level Programmers’ Model).

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

**C, bit [2]**

Cacheability control, for data accesses.

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All data access to Normal memory from EL3, and all Normal memory accesses to the EL3 translation tables, are Non-cacheable for all levels of data and unified cache.</td>
</tr>
</tbody>
</table>
| 0b1 | This control has no effect on the Cacheability of:  
• Data access to Normal memory from EL3.  
• Normal memory accesses to the EL3 translation tables. |

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

In a system where the PE resets into EL3, this field resets to 0.
A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL3.

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Alignment fault checking disabled when executing at EL3. Instructions</td>
</tr>
<tr>
<td></td>
<td>that load or store one or more registers, other than load/store exclusive</td>
</tr>
<tr>
<td></td>
<td>and load-acquire/store-release, do not check that the address being</td>
</tr>
<tr>
<td></td>
<td>accessed is aligned to the size of the data element(s) being accessed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Alignment fault checking enabled when executing at EL3. All instructions</td>
</tr>
<tr>
<td></td>
<td>that load or store one or more registers have an alignment check that</td>
</tr>
<tr>
<td></td>
<td>the address being accessed is aligned to the size of the data element(s)</td>
</tr>
<tr>
<td></td>
<td>being accessed. If this check fails it causes an Alignment fault, which</td>
</tr>
<tr>
<td></td>
<td>is taken as a Data Abort exception.</td>
</tr>
</tbody>
</table>

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL3 stage 1 address translation. Possible values of this bit are:

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL3 stage 1 address translation disabled. See the SCTLR_EL3.I field</td>
</tr>
<tr>
<td></td>
<td>for the behavior of instruction accesses to Normal memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL3 stage 1 address translation enabled.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to 0.

Accessing the SCTLR_EL3

Accesses to this register use the following encodings:

**MRS <Xt>, SCTLR_EL3**

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return SCTLR_EL3;
```

**MSR SCTLR_EL3, <Xt>**

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return SCTLR_EL3;
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCTLR_EL3 = X[t];
The SCXTNUM_EL0 characteristics are:

**Purpose**

Provides a number that can be used to separate out different context numbers with the EL0 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

**Configuration**

This register is present only when ARMv8.0-CSV2 is implemented. Otherwise, direct accesses to SCXTNUM_EL0 are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

SCXTNUM_EL0 is a 64-bit register.

**Field descriptions**

The SCXTNUM_EL0 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Software Context Number**

Software Context Number. A number to identify the context within the EL0 exception level.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SCXTNUM_EL0**

Accesses to this register use the following encodings:

**MRS <Xt>, SCXTNUM_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.TSCXT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    AArch64.SystemAccessTrap(EL1, 0x18);
  endif
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && SCTLR_EL1.TSCXT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return SCXTNUM_EL0;
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.EnSCXT == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.SCXTNUM_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.TSCXT == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return SCXTNUM_EL0;
  endif
elsif PSTATE.EL == EL3 then
  if HaveEL(EL3) && !ELUsingAAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return SCXTNUM_EL0;
  endif
else
  if PSTATE.EL == EL3 then
    return SCXTNUM_EL0;
  endif
end if

MSR SCXTNUM_EL0, <Xt>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && SCTLR_EL1.TSCXT == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && SCTLR_EL1.TSCXT == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      AArch64.SystemAccessTrap(EL1, 0x18);
    end
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL0 == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      SCXTNUM_EL0 = X[t];
    end
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EnSCXT == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL0 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      SCXTNUM_EL0 = X[t];
    end
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      SCXTNUM_EL0 = X[t];
    end
  elsif PSTATE.EL == EL3 then
    SCXTNUM_EL0 = X[t];
  end
SCXTNUM_EL1, EL1 Read/Write Software Context Number

The SCXTNUM_EL1 characteristics are:

**Purpose**

Provides a number that can be used to separate out different context numbers with the EL1 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

**Configuration**

This register is present only when ARMv8.0-CSV2 is implemented. Otherwise, direct accesses to SCXTNUM_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SCXTNUM_EL1 is a 64-bit register.

**Field descriptions**

The SCXTNUM_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

Software Context Number. A number to identify the context within the EL1 exception level. This field resets to an architecturally UNKNOWN value.

**Accessing the SCXTNUM_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, SCXTNUM_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.SCXTNUM_EL1 == '111' then
        return NVMem[0x188];
    else
        return SCXTNUM_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            SCXTNUM_EL2 = X[t];
        else
            SCXTNUM_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        SCXTNUM_EL1 = X[t];
MRS SCXTNUM_EL12

MRS <Xt>, SCXTNUM_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b00000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x188];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsf PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      SCXTNUM_EL1 = X[t];
    else
      UNDEFINED;
elsf PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    SCXTNUM_EL1 = X[t];
  else
    UNDEFINED;

MSR SCXTNUM_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x188] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsf PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      SCXTNUM_EL1 = X[t];
    else
      UNDEFINED;
elsf PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    SCXTNUM_EL1 = X[t];
  else
    UNDEFINED;
SCXTNUM_EL2, EL2 Read/Write Software Context Number

The SCXTNUM_EL2 characteristics are:

**Purpose**

Provides a number that can be used to separate out different context numbers with the EL2 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

**Configuration**

This register is present only when ARMv8.0-CSV2 is implemented. Otherwise, direct accesses to SCXTNUM_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SCXTNUM_EL2 is a 64-bit register.

**Field descriptions**

The SCXTNUM_EL2 bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 63     | 62     | 61     | 60     | 59     | 58     | 57     | 56     | 55     | 54     | 53     | 52     | 51     | 50     | 49     | 48     | 47     | 46     | 45     | 44     | 43     | 42     | 41     | 40     | 39     | 38     | 37     | 36     | 35     | 34     | 33     | 32     |
| Software Context Number | Software Context Number |
| 31     | 30     | 29     | 28     | 27     | 26     | 25     | 24     | 23     | 22     | 21     | 20     | 19     | 18     | 17     | 16     | 15     | 14     | 13     | 12     | 11     | 10     | 9      | 8      | 7      | 6      | 5      | 4      | 3      | 2      | 1      | 0      |

**Bits [63:0]**

Software Context Number. A number to identify the context within the EL2 exception level.

This field resets to an architecturally UNKNOWN value.

**Accessing the SCXTNUM_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic SCXTNUM_EL2 or SCXTNUM_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, SCXTNUM_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL2 = X[t];
MRS SCXTNUM_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

eif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x188];
    else
        return SCXTNUM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return SCXTNUM_EL2;
    else
        return SCXTNUM_EL1;
elsif PSTATE.EL == EL3 then
    return SCXTNUM_EL1;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.EnSCXT == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.SCXTNUM_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.EnSCXT == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x188] = X[t];
  else
    SCXTNUM_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.EnSCXT == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    SCXTNUM_EL2 = X[t];
  else
    SCXTNUM_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  SCXTNUM_EL1 = X[t];
SCXTNUM_EL3, EL3 Read/Write Software Context Number

The SCXTNUM_EL3 characteristics are:

**Purpose**

Provides a number that can be used to separate out different context numbers with the EL3 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

**Configuration**

This register is present only when HaveEL(EL3) and ARMv8.0-CSV2 is implemented. Otherwise, direct accesses to SCXTNUM_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SCXTNUM_EL3 is a 64-bit register.

**Field descriptions**

The SCXTNUM_EL3 bit assignments are:

|   | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|   | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

<table>
<thead>
<tr>
<th>Software Context Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Context Number</td>
</tr>
</tbody>
</table>

**Software Context Number**

A number to identify the context within the EL3 exception level. This field resets to an architecturally UNKNOWN value.

**Accessing the SCXTNUM_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, SCXTNUM_EL3

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b11 & 0b110 & 0b1101 & 0b0000 & 0b111 \\
\hline
\end{array}
\]

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return SCXTNUM_EL3;
MSR SCXTNUM_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL3 = X[t];
SDER32_EL2, AArch32 Secure Debug Enable Register

The SDER32_EL2 characteristics are:

**Purpose**

Allows access to the AArch32 register SDER from Secure EL2 and EL3 only.

**Configuration**

This register is present only when ARMv8.4-SecEL2 is implemented and EL1 supports AArch32. Otherwise, direct accesses to SDER32_EL2 are **UNDEFINED**.

This register is ignored by the PE when one or more of the following are true:

- The PE is in Non-secure state.
- EL1 is using AArch64.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

SDER32_EL2 is a 64-bit register.

**Field descriptions**

The SDER32_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>31</td>
<td>SUNIDEN</td>
<td>Secure User Non-Invasive Debug Enable.</td>
</tr>
<tr>
<td>30</td>
<td>SUIDEN</td>
<td>Secure User Invasive Debug Enable.</td>
</tr>
</tbody>
</table>

- **SUNIDEN**, bit [1]

  Secure User Non-Invasive Debug Enable.

<table>
<thead>
<tr>
<th>SUNIDEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This bit does not affect Performance Monitors event counting at Secure EL0.</td>
</tr>
<tr>
<td>1</td>
<td>If EL1 is using AArch32, Performance Monitors event counting is allowed in Secure EL0.</td>
</tr>
</tbody>
</table>

  On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

- **SUIDEN**, bit [0]

  Secure User Invasive Debug Enable.
**SDER32_EL2, AArch32 Secure Debug Enable Register**

<table>
<thead>
<tr>
<th>SUIDEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit does not affect the generation of debug exceptions at Secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the SDER32_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, SDER32_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SDER32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SDER32_EL2 = X[t];
```

**MSR SDER32_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SDER32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SDER32_EL2 = X[t];
```
SDER32_EL3, AArch32 Secure Debug Enable Register

The SDER32_EL3 characteristics are:

**Purpose**

Allows access to the AArch32 register SDER from AArch64 state only. Its value has no effect on execution in AArch64 state.

**Configuration**

AArch64 System register SDER32_EL3 bits [31:0] are architecturally mapped to AArch32 System register SDER[31:0].

This register is present only when HaveEL(EL3) and EL1 supports AArch32. Otherwise, direct accesses to SDER32_EL3 are UNDEFINED.

This register is ignored by the PE when when one or more of the following are true:

- The PE is in Non-secure state.
- EL1 is using AArch64.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SDER32_EL3 is a 64-bit register.

**Field descriptions**

The SDER32_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>RES0</td>
<td>SUIDEN</td>
<td>SUIDEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:2]**

Reserved, RES0.

**SUNIDEN, bit [1]**

Secure User Non-Invasive Debug Enable.

<table>
<thead>
<tr>
<th>SUNIDEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit does not affect Performance Monitors event counting at Secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL1 is using AArch32, Performance Monitors event counting is allowed in Secure EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**SUIDEN, bit [0]**

Secure User Invasive Debug Enable.
### SUIDEN

<table>
<thead>
<tr>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit does not affect the generation of debug exceptions at Secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

---

## Accessing the SDER32_EL3

Accesses to this register use the following encodings:

**MRS <Xt>, SDER32_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return SDER32_EL3;
```

**MSR SDER32_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  SDER32_EL3 = X[t];
```
SP_EL0, Stack Pointer (EL0)

The SP_EL0 characteristics are:

**Purpose**

Holds the stack pointer associated with EL0. At higher Exception levels, this is used as the current stack pointer when the value of $SPSel$.SP is 0.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SP_EL0 is a 64-bit register.

**Field descriptions**

The SP_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Stack pointer | Stack pointer |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**Bits [63:0]**

Stack pointer:

This field resets to an architecturally UNKNOWN value.

**Accessing the SP_EL0**

When the value of PSTATE.SP is 0, this register is accessible at all Exception levels as the current stack pointer.

Accesses to this register use the following encodings:

**MRS <Xt>, SP_EL0**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if PSTATE.SP == '0' then
    UNDEFINED;
  else
    return SP_EL0;
elsif PSTATE.EL == EL2 then
  if PSTATE.SP == '0' then
    UNDEFINED;
  else
    return SP_EL0;
elsif PSTATE.EL == EL3 then
  if PSTATE.SP == '0' then
    UNDEFINED;
  else
    return SP_EL0;

MSR SP_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if PSTATE.SP == '0' then
    UNDEFINED;
  else
    SP_EL0 = X[t];
elsif PSTATE.EL == EL2 then
  if PSTATE.SP == '0' then
    UNDEFINED;
  else
    SP_EL0 = X[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.SP == '0' then
    UNDEFINED;
  else
    SP_EL0 = X[t];
SP_EL1, Stack Pointer (EL1)

The SP_EL1 characteristics are:

**Purpose**

Holds the stack pointer associated with EL1. When executing at EL1, the value of `SPSel.SP` determines the current stack pointer:

<table>
<thead>
<tr>
<th><code>SPSel.SP</code></th>
<th>Current stack pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SP_EL0</td>
</tr>
<tr>
<td>0b1</td>
<td>SP_EL1</td>
</tr>
</tbody>
</table>

**Configuration**

RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**

SP_EL1 is a 64-bit register.

**Field descriptions**

The SP_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>62</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>31</td>
<td>Stack pointer</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Stack pointer.

This field resets to an architecturally `UNKNOWN` value.

**Accessing the SP_EL1**

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of `SPSel.SP` is 1, this register is also accessible at EL1 as the current stack pointer.

**Note**

When the value of `SPSel.SP` is 0, SP_EL0 is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings:

**MRS <Xt>, SP_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x240];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return SP_EL1;
elsif PSTATE.EL == EL3 then
  return SP_EL1;

MSR SP_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x240] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  SP_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  SP_EL1 = X[t];
The SP_EL2 characteristics are:

**Purpose**

Holds the stack pointer associated with EL2. When executing at EL2, the value of `SPSel.SP` determines the current stack pointer:

<table>
<thead>
<tr>
<th><code>SPSel.SP</code></th>
<th>Current stack pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><code>SP_EL0</code></td>
</tr>
<tr>
<td>0b1</td>
<td><code>SP_EL2</code></td>
</tr>
</tbody>
</table>

**Configuration**

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**

SP_EL2 is a 64-bit register.

**Field descriptions**

The SP_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>Stack pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>Stack pointer</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Stack pointer.

This field resets to an architecturally `UNKNOWN` value.

**Accessing the SP_EL2**

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of `SPSel.SP` is 1, this register is also accessible at EL2 as the current stack pointer.

**Note**

When the value of `SPSel.SP` is 0, `SP_EL0` is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings:

**MRS <Xt>, SP_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SP_EL2;

MSR SP_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SP_EL2 = X[t];
SP_EL3, Stack Pointer (EL3)

The SP_EL3 characteristics are:

**Purpose**

Holds the stack pointer associated with EL3. When executing at EL3, the value of SPSel.SP determines the current stack pointer:

<table>
<thead>
<tr>
<th>SPSel.SP</th>
<th>Current stack pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SP_EL0</td>
</tr>
<tr>
<td>0b1</td>
<td>SP_EL3</td>
</tr>
</tbody>
</table>

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to SP_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SP_EL3 is a 64-bit register.

**Field descriptions**

The SP_EL3 bit assignments are:

```
<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Stack pointer</td>
</tr>
<tr>
<td>32</td>
<td>Stack pointer</td>
</tr>
</tbody>
</table>
```

**Bits [63:0]**

Stack pointer.

This field resets to an architecturally UNKNOWN value.

**Accessing the SP_EL3**

This register is not accessible using MRS and MSR instructions.

When the value of SPSel.SP is 1, this register is accessible at EL3 as the current stack pointer.

**Note**

When the value of SPSel.SP is 0, SP_EL0 is used as the current stack pointer at all Exception levels.
SPSel, Stack Pointer Select

The SPSel characteristics are:

**Purpose**

Allows the Stack Pointer to be selected between SP_EL0 and SP_ELx.

**Configuration**

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSel is a 64-bit register.

**Field descriptions**

The SPSel bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| RES0   | RES0   | SP     |

**Bits [63:1]**

Reserved, RES0.

**SP, bit [0]**

Stack pointer to use. Possible values of this bit are:

<table>
<thead>
<tr>
<th>SP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Use SP_EL0 at all Exception levels.</td>
</tr>
<tr>
<td>0b1</td>
<td>Use SP_ELx for Exception level ELx.</td>
</tr>
</tbody>
</table>

This field resets to 1.

**Accessing the SPSel**

For details on the operation of the MSR (immediate) accessor, see MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

MRS <Xt>, SPSel

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
return Zeros(63):PSTATE.SP;
elsif PSTATE.EL == EL2 then
return Zeros(63):PSTATE.SP;
elsif PSTATE.EL == EL3 then
return Zeros(63):PSTATE.SP;

MSR SPSel, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
  PSTATE.SP = X[t]<0>;
elsif PSTATE.EL == EL2 then
  PSTATE.SP = X[t]<0>;
elsif PSTATE.EL == EL3 then
  PSTATE.SP = X[t]<0>;

MSR SPSel, #<imm>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>
SPSR_abt, Saved Program Status Register (Abort mode)

The SPSR_abt characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to Abort mode.

**Configuration**

AArch64 System register SPSR_abt bits [31:0] are architecturally mapped to AArch32 System register SPSR_abt[31:0].

If EL1 does not support execution in AArch32 state, this register is RES0 from EL2 and EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_abt is a 64-bit register.

**Field descriptions**

The SPSR_abt bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Abort mode, and copied to PSTATE.Z on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Abort mode, and copied to PSTATE.C on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.
V, bit [28]
Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Abort mode, and copied to PSTATE.V on executing an exception return operation in Abort mode.
This field resets to an architecturally UNKNOWN value.

Q, bit [27]
Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Abort mode, and copied to PSTATE.Q on executing an exception return operation in Abort mode.
This field resets to an architecturally UNKNOWN value.

IT[1:0], bits [26:25]
If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Abort mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Abort mode.
On executing an exception return operation in Abort mode SPSR_abt.IT must contain a value that is valid for the instruction being returned to.
This field resets to an architecturally UNKNOWN value.

J, bit [24]
RES0.
In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]
When ARMv8.0-SSBS is implemented:
Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Abort mode, and copied to PSTATE.SSBS on executing an exception return operation in Abort mode.
This field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

PAN, bit [22]
When ARMv8.1-PAN is implemented:
Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Abort mode, and copied to PSTATE.PAN on executing an exception return operation in Abort mode.
This field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.
DIT, bit [21]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Abort mode, and copied to PSTATE.DIT on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, **RES0**.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Abort mode, and copied to PSTATE.IL on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Abort mode, and copied to PSTATE.GE on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Abort mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in Abort mode.

SPSR_abt.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Abort mode, and copied to PSTATE.E on executing an exception return operation in Abort mode.

If the implementation does not support big-endian operation, SPSR_abt.E is **RES0**. If the implementation does not support little-endian operation, SPSR_abt.E is **RES1**. On executing an exception return operation in Abort mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_abt.E is **RES0**, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_abt.E is **RES1**.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Abort mode, and copied to PSTATE.A on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Abort mode, and copied to PSTATE.I on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.
F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Abort mode, and copied to PSTATE.F on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Abort mode, and copied to PSTATE.T on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Abort mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Abort mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_abt.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Abort mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**Accessing the SPSR_abt**

Accesses to this register use the following encodings:

**MRS <Xt>, SPSR_abt**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_abt;
elsif PSTATE.EL == EL3 then
    return SPSR_abt;

**MSR SPSR_abt, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0
    UNDEFINED;
elsif PSTATE.EL == EL1
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2
    SPSR_abt = X[t];
elsif PSTATE.EL == EL3
    SPSR_abt = X[t];
SPSR_EL1, Saved Program Status Register (EL1)

The SPSR_EL1 characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to EL1.

**Configuration**

AArch64 System register SPSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register SPSR_svc[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_EL1 is a 64-bit register.

**Field descriptions**

The SPSR_EL1 bit assignments are:

**When exception taken from AArch32 state:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.
V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL1, and copied to PSTATE.Q on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to EL1, and copied to PSTATE.IT[1:0] on executing an exception return operation in EL1.

On executing an exception return operation in EL1 SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**DIT, bit [24]**

*When* ARMv8.4-DIT *is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**SSBS, bit [23]**

*When* ARMv8.0-SSBS *is implemented:*

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

*When* ARMv8.1-PAN *is implemented:*

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**SS, bit [21]**

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL1, and copied to PSTATE.GE on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to EL1, and copied to PSTATE.IT[7:2] on executing an exception return operation in EL1.

SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to EL1, and copied to PSTATE.E on executing an exception return operation in EL1.

If the implementation does not support big-endian operation, SPSR_EL1.E is RES0. If the implementation does not support little-endian operation, SPSR_EL1.E is RES1. On executing an exception return operation in EL1, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL1.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL1.E is RES1.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.
FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**T, bit [5]**

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL1, and copied to PSTATE.T on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**M[4], bit [4]**

Execution state. Set to 0b1, the value of PSTATE.nRW, on taking an exception to EL1 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>AArch32 execution state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**M[3:0], bits [3:0]**

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL1, and copied to PSTATE.M[3:0] on executing an exception return operation in EL1.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>User.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b1111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in ‘Illegal return events from AArch64 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**When exception taken from AArch64 state:**

An exception return from EL1 using AArch64 makes SPSR_EL1 become **UNKNOWN**.

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.
Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When ARMv8.5-MemTag is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL1, and copied to PSTATE.TCO on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When ARMv8.2-UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL1, and copied to PSTATE.UAO on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**PAN, bit [22]**

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**SS, bit [21]**

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

**Bits [19:13]**

Reserved, RES0.

**SSBS, bit [12]**

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**BTYPE, bits [11:10]**

When ARMv8.5-BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL1, and copied to PSTATE.BTYPE on executing an exception return operation in EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.
D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL1, and copied to PSTATE.D on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

This field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on taking an exception to EL1 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch64 execution state.</td>
</tr>
</tbody>
</table>

If AArch32 is not supported at any Exception level, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL0t.</td>
</tr>
<tr>
<td>0b0100</td>
<td>EL1t.</td>
</tr>
<tr>
<td>0b0101</td>
<td>EL1h.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1 and copied to PSTATE.EL on executing an exception return operation in EL1.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL1 and copied to PSTATE.SP on executing an exception return operation in EL1.
This field resets to an architecturally **UNKNOWN** value.

## Accessing the SPSR_EL1

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL3 using the mnemonic SPSR_EL1 or SPSR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

### MRS <Xt>, SPSR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x160];
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SPSR_EL2;
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL3 then
    return SPSR_EL1;

### MSR SPSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x160] = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SPSR_EL2 = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t];

### MRS <Xt>, SPSR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x160];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return SPSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return SPSR_EL1;
    else
        UNDEFINED;

MSR SPSR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x160] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        SPSR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        SPSR_EL1 = X[t];
    else
        UNDEFINED;

MRS <Xt>, SPSR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return SPSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_EL2;
elsif PSTATE.EL == EL3 then
    return SPSR_EL2;
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        SPSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    SPSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL2 = X[t];

MSR SPSR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
SPSR_EL2, Saved Program Status Register (EL2)

The SPSR_EL2 characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to EL2.

**Configuration**

AArch64 System register SPSR_EL2 bits [31:0] are architecturally mapped to AArch32 System register SPSR_hyp[31:0].

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

SPSR_EL2 is a 64-bit register.

**Field descriptions**

The SPSR_EL2 bit assignments are:

**When exception taken from AArch32 state:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>62</td>
<td><strong>N</strong></td>
</tr>
<tr>
<td>61</td>
<td><strong>Z</strong></td>
</tr>
<tr>
<td>60</td>
<td><strong>C</strong></td>
</tr>
<tr>
<td>59</td>
<td><strong>V</strong></td>
</tr>
<tr>
<td>58</td>
<td><strong>Q</strong></td>
</tr>
<tr>
<td>57</td>
<td><strong>IT[1:0]</strong></td>
</tr>
<tr>
<td>56</td>
<td><strong>SSB</strong></td>
</tr>
<tr>
<td>55</td>
<td><strong>SS</strong></td>
</tr>
<tr>
<td>54</td>
<td><strong>PAN</strong></td>
</tr>
<tr>
<td>53</td>
<td><strong>IL</strong></td>
</tr>
<tr>
<td>52</td>
<td><strong>GE</strong></td>
</tr>
<tr>
<td>51</td>
<td><strong>E</strong></td>
</tr>
<tr>
<td>50</td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>49</td>
<td><strong>I</strong></td>
</tr>
<tr>
<td>48</td>
<td><strong>F</strong></td>
</tr>
<tr>
<td>47</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>46</td>
<td><strong>M[4]</strong></td>
</tr>
<tr>
<td>45</td>
<td><strong>M[3:0]</strong></td>
</tr>
<tr>
<td>44</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>43</td>
<td><strong>E</strong></td>
</tr>
<tr>
<td>42</td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>41</td>
<td><strong>I</strong></td>
</tr>
<tr>
<td>40</td>
<td><strong>F</strong></td>
</tr>
<tr>
<td>39</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>38</td>
<td><strong>M[4]</strong></td>
</tr>
<tr>
<td>37</td>
<td><strong>M[3:0]</strong></td>
</tr>
<tr>
<td>36</td>
<td><strong>T</strong></td>
</tr>
<tr>
<td>35</td>
<td><strong>E</strong></td>
</tr>
<tr>
<td>34</td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>33</td>
<td><strong>I</strong></td>
</tr>
<tr>
<td>32</td>
<td><strong>F</strong></td>
</tr>
</tbody>
</table>

An exception return from EL2 using AArch64 makes SPSR_EL2 become **UNKNOWN**.

**Bits [63:32]**

Reserved, **RES0**.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL2, and copied to PSTATE.N on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL2, and copied to PSTATE.Z on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL2, and copied to PSTATE.C on executing an exception return operation in EL2.
This field resets to an architecturally **UNKNOWN** value.

**V, bit [28]**

Overflow Condition flag. Set to the value of \text{PSTATE.V} on taking an exception to EL2, and copied to \text{PSTATE.V} on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**Q, bit [27]**

Overflow or saturation flag. Set to the value of \text{PSTATE.Q} on taking an exception to EL2, and copied to \text{PSTATE.Q} on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of \text{PSTATE.IT[1:0]} on taking an exception to EL2, and copied to \text{PSTATE.IT[1:0]} on executing an exception return operation in EL2.

On executing an exception return operation in EL2 \text{SPSR_EL2.IT} must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**DIT, bit [24]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of \text{PSTATE.DIT} on taking an exception to EL2, and copied to \text{PSTATE.DIT} on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, \text{RES0}.

**SSBS, bit [23]**

*When ARMv8.0-SSBS is implemented:*

Speculative Store Bypass. Set to the value of \text{PSTATE.SSBS} on taking an exception to EL2, and copied to \text{PSTATE.SSBS} on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, \text{RES0}.

**PAN, bit [22]**

*When ARMv8.1-PAN is implemented:*

Privileged Access Never. Set to the value of \text{PSTATE.PAN} on taking an exception to EL2, and copied to \text{PSTATE.PAN} on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**SS, bit [21]**

Software Step. Set to the value of PSTATE.SS on taking an exception to EL2, and conditionally copied to PSTATE.SS on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL2, and copied to PSTATE.IL on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL2, and copied to PSTATE.GE on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to EL2, and copied to PSTATE.IT[7:2] on executing an exception return operation in EL2.

SPSR_EL2.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to EL2, and copied to PSTATE.E on executing an exception return operation in EL2.

If the implementation does not support big-endian operation, SPSR_EL2.E is RES0. If the implementation does not support little-endian operation, SPSR_EL2.E is RES1. On executing an exception return operation in EL2, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL2.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL2.E is RES1.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL2, and copied to PSTATE.A on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL2, and copied to PSTATE.I on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.
**F, bit [6]**

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL2, and copied to PSTATE.F on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**T, bit [5]**

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL2, and copied to PSTATE.T on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**M[4], bit [4]**

Execution state. Set to 0b1, the value of PSTATE.nRW, on taking an exception to EL2 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL2.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>AArch32 execution state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**M[3:0], bits [3:0]**

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL2, and copied to PSTATE.M[3:0] on executing an exception return operation in EL2.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>User.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b1111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_EL2.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL2 is an illegal return event, as described in ‘Illegal return events from AArch64 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**When exception taken from AArch64 state:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| N | Z | C | V | RES0 | TCOD | DIT | UAOP | PAN | SS | IL | RES0 | SSBS | BTYPE | D | A | I | F | RES0 | M[4] | M[3:0] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return from EL2 using AArch64 makes SPSR_EL2 become **UNKNOWN**.

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL2, and copied to PSTATE.N on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.
**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL2, and copied to PSTATE.Z on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL2, and copied to PSTATE.C on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL2, and copied to PSTATE.V on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**Bits [27:26]**

Reserved, RES0.

**TCO, bit [25]**

When ARMv8.5-MemTag is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL2, and copied to PSTATE.TCO on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**DIT, bit [24]**

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL2, and copied to PSTATE.DIT on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**UAO, bit [23]**

When ARMv8.2-UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL2, and copied to PSTATE.UAO on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**PAN, bit [22]**  
When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL2, and copied to PSTATE.PAN on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**SS, bit [21]**

Software Step. Set to the value of PSTATE.SS on taking an exception to EL2, and conditionally copied to PSTATE.SS on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL2, and copied to PSTATE.IL on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

**Bits [19:13]**

Reserved, RES0.

**SSBS, bit [12]**  
When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL2, and copied to PSTATE.SSBS on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**BTYPE, bits [11:10]**  
When ARMv8.5-BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL2, and copied to PSTATE.BTYPE on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.
D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL2, and copied to PSTATE.D on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL2, and copied to PSTATE.A on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL2, and copied to PSTATE.I on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL2, and copied to PSTATE.F on executing an exception return operation in EL2.

This field resets to an architecturally **UNKNOWN** value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on taking an exception to EL2 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL2.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch64 execution state.</td>
</tr>
</tbody>
</table>

If AArch32 is not supported at any Exception level, this bit is RES0.

This field resets to an architecturally **UNKNOWN** value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL0t.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL0h.</td>
</tr>
<tr>
<td>0b0100</td>
<td>EL1t.</td>
</tr>
<tr>
<td>0b0101</td>
<td>EL1h.</td>
</tr>
<tr>
<td>0b1000</td>
<td>EL2t.</td>
</tr>
<tr>
<td>0b1001</td>
<td>EL2h.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_EL2.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL2 is an illegal return event, as described in 'Illegal return events from AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL2 and copied to PSTATE.EL on executing an exception return operation in EL2.
- M[1] is unused and is 0 for all non-reserved values.
• M[0] is set to the value of PSTATE.SP on taking an exception to EL2 and copied to PSTATE.SP on executing an exception return operation in EL2

This field resets to an architecturally **UNKNOWN** value.

## Accessing the SPSR_EL2

When `HCR_EL2.E2H` is 1, without explicit synchronization, access from EL2 using the mnemonic `SPSR_EL2` or `SPSR_EL1` are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

### MRS <Xt>, SPSR_EL2

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return SPSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        return SPSR_EL2;
    elsif PSTATE.EL == EL3 then
        return SPSR_EL2;
endif
```

### MSR SPSR_EL2, <Xt>

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        SPSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        SPSR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        SPSR_EL2 = X[t];
endif
```

### MRS <Xt>, SPSR_EL1

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x160];
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SPSR_EL2;
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL3 then
    return SPSR_EL1;

MSR SPSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x160] = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SPSR_EL2 = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t];
SPSR_EL3, Saved Program Status Register (EL3)

The SPSR_EL3 characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to EL3.

**Configuration**

AArch64 System register SPSR_EL3 bits [31:0] can be mapped to AArch32 System register SPSR_mon[31:0], but this is not architecturally mandated.

This register is present only when HaveEL(EL3). Otherwise, direct accesses to SPSR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_EL3 is a 64-bit register.

**Field descriptions**

The SPSR_EL3 bit assignments are:

**When exception taken from AArch32 state:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

An exception return from EL3 using AArch64 makes SPSR_EL1 become UNKNOWN.

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL3, and copied to PSTATE.N on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL3, and copied to PSTATE.Z on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL3, and copied to PSTATE.C on executing an exception return operation in EL3.
This field resets to an architecturally **UNKNOWN** value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL3, and copied to PSTATE.V on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**Q, bit [27]**

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL3, and copied to PSTATE.Q on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to EL3, and copied to PSTATE.IT[1:0] on executing an exception return operation in EL3.

On executing an exception return operation in EL3 SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**DIT, bit [24]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL3, and copied to PSTATE.DIT on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**SSBS, bit [23]**

*When ARMv8.0-SSBS is implemented:*

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL3, and copied to PSTATE.SSBS on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

*When ARMv8.1-PAN is implemented:*

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL3, and copied to PSTATE.PAN on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**SS, bit [21]**

Software Step. Set to the value of PSTATE.SS on taking an exception to EL3, and conditionally copied to PSTATE.SS on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL3, and copied to PSTATE.IL on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL3, and copied to PSTATE.GE on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to EL3, and copied to PSTATE.IT[7:2] on executing an exception return operation in EL3.

SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to EL3, and copied to PSTATE.E on executing an exception return operation in EL3.

If the implementation does not support big-endian operation, SPSR_EL1.E is RES0. If the implementation does not support little-endian operation, SPSR_EL1.E is RES1. On executing an exception return operation in EL3, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL1.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL1.E is RES1.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL3, and copied to PSTATE.A on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL3, and copied to PSTATE.I on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.
F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL3, and copied to PSTATE.F on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL3, and copied to PSTATE.T on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to 0b1, the value of PSTATE.nRW, on taking an exception to EL3 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL3.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>AArch32 execution state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL3, and copied to PSTATE.M[3:0] on executing an exception return operation in EL3.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>User.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Monitor.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b1111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL3 is an illegal return event, as described in ‘Illegal return events from AArch64 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:

An exception return from EL3 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL3, and copied to PSTATE.N on executing an exception return operation in EL3.
This field resets to an architecturally **UNKNOWN** value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL3, and copied to PSTATE.Z on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL3, and copied to PSTATE.C on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL3, and copied to PSTATE.V on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

**Bits [27:26]**

Reserved, RES0.

**TCO, bit [25]**

*When ARMv8.5-MemTag is implemented:*

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL3, and copied to PSTATE.TCO on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**DIT, bit [24]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL3, and copied to PSTATE.DIT on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**UAO, bit [23]**

*When ARMv8.2-UAO is implemented:*

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL3, and copied to PSTATE.UAO on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL3, and copied to PSTATE.PAN on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL3, and conditionally copied to PSTATE.SS on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL3, and copied to PSTATE.IL on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

Bits [19:13]

Reserved, RES0.

SSBS, bit [12]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL3, and copied to PSTATE.SSBS on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When ARMv8.5-BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL3, and copied to PSTATE.BTYPE on executing an exception return operation in EL3.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL3, and copied to PSTATE.D on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL3, and copied to PSTATE.A on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL3, and copied to PSTATE.I on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL3, and copied to PSTATE.F on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on taking an exception to EL3 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL3.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch64 execution state.</td>
</tr>
</tbody>
</table>

If AArch32 is not supported at any Exception level, this bit is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL0t</td>
</tr>
<tr>
<td>0b0100</td>
<td>EL1t</td>
</tr>
<tr>
<td>0b1001</td>
<td>EL1h</td>
</tr>
<tr>
<td>0b1000</td>
<td>EL2t</td>
</tr>
<tr>
<td>0b1001</td>
<td>EL2h</td>
</tr>
<tr>
<td>0b1100</td>
<td>EL3t</td>
</tr>
<tr>
<td>0b1101</td>
<td>EL3h</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_EL3.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL3 is an illegal return event, as described in 'Illegal return events from AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The bits in this field are interpreted as follows:
• M[3:2] is set to the value of PSTATE.EL on taking an exception to EL3 and copied to PSTATE.EL on executing an exception return operation in EL3.
• M[1] is unused and is 0 for all non-reserved values.
• M[0] is set to the value of PSTATE.SP on taking an exception to EL3 and copied to PSTATE.SP on executing an exception return operation in EL3.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the SPSR_EL3

Accesses to this register use the following encodings:

**MRS <Xt>, SPSR_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return SPSR_EL3;
```

**MSR SPSR_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b11</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  SPSR_EL3 = X[t];
```
SPSR_fiq, Saved Program Status Register (FIQ mode)

The SPSR_fiq characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to FIQ mode.

**Configuration**

AArch64 System register SPSR_fiq bits [31:0] are architecturally mapped to AArch32 System register `SPSR_fiq[31:0]`. If EL1 does not support execution in AArch32 state, this register is Res0 from EL2 and EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_fiq is a 64-bit register.

**Field descriptions**

The SPSR_fiq bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, Res0</td>
</tr>
<tr>
<td>31</td>
<td>N, Negative Condition flag</td>
</tr>
<tr>
<td>30</td>
<td>Z, Zero Condition flag</td>
</tr>
<tr>
<td>29</td>
<td>C, Carry Condition flag</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, Res0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to FIQ mode, and copied to PSTATE.N on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to FIQ mode, and copied to PSTATE.Z on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to FIQ mode, and copied to PSTATE.C on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.
V, bit [28]
Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to FIQ mode, and copied to PSTATE.V on executing an exception return operation in FIQ mode.
This field resets to an architecturally "UNKNOWN" value.

Q, bit [27]
Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to FIQ mode, and copied to PSTATE.Q on executing an exception return operation in FIQ mode.
This field resets to an architecturally "UNKNOWN" value.

IT[1:0], bits [26:25]
If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to FIQ mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in FIQ mode.
On executing an exception return operation in FIQ mode SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.
This field resets to an architecturally "UNKNOWN" value.

J, bit [24]
RES0.
In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]
When Armv8.0-SSBS is implemented:
Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to FIQ mode, and copied to PSTATE.SSBS on executing an exception return operation in FIQ mode.
This field resets to an architecturally "UNKNOWN" value.

Otherwise:
Reserved, RES0.

PAN, bit [22]
When Armv8.1-PAN is implemented:
Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to FIQ mode, and copied to PSTATE.PAN on executing an exception return operation in FIQ mode.
This field resets to an architecturally "UNKNOWN" value.

Otherwise:
Reserved, RES0.
DIT, bit [21]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to FIQ mode, and copied to PSTATE.DIT on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to FIQ mode, and copied to PSTATE.IL on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to FIQ mode, and copied to PSTATE.GE on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to FIQ mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in FIQ mode.

SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to FIQ mode, and copied to PSTATE.E on executing an exception return operation in FIQ mode.

If the implementation does not support big-endian operation, SPSR_fiq.E is RES0. If the implementation does not support little-endian operation, SPSR_fiq.E is RES1. On executing an exception return operation in FIQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_fiq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_fiq.E is RES1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to FIQ mode, and copied to PSTATE.A on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to FIQ mode, and copied to PSTATE.I on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.
F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to FIQ mode, and copied to PSTATE.F on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to FIQ mode, and copied to PSTATE.T on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to FIQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in FIQ mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_fiq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in FIQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_fiq**

Accesses to this register use the following encodings:

**MRS <Xt>, SPSR_fiq**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == ‘1’ then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  return SPSR_fiq;
elseif PSTATE.EL == EL3 then
  return SPSR_fiq;

**MSR SPSR_fiq, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  SPSR_fiq = X[t];
elsif PSTATE.EL == EL3 then
  SPSR_fiq = X[t];
SPSR_irq, Saved Program Status Register (IRQ mode)

The SPSR_irq characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to IRQ mode.

**Configuration**

AArch64 System register SPSR_irq bits [31:0] are architecturally mapped to AArch32 System register SPSR_irq[31:0]. If EL1 does not support execution in AArch32 state, this register is RES0 from EL2 and EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_irq is a 64-bit register.

**Field descriptions**

The SPSR_irq bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to IRQ mode, and copied to PSTATE.N on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to IRQ mode, and copied to PSTATE.Z on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to IRQ mode, and copied to PSTATE.C on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.
V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to IRQ mode, and copied to PSTATE.V on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to IRQ mode, and copied to PSTATE.Q on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

IT[1:0], bits [26:25]

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to IRQ mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in IRQ mode.

On executing an exception return operation in IRQ mode SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to IRQ mode, and copied to PSTATE.SSBS on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to IRQ mode, and copied to PSTATE.PAN on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
**DIT, bit [21]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to IRQ mode, and copied to PSTATE.DIT on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, **RES0**.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to IRQ mode, and copied to PSTATE.IL on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to IRQ mode, and copied to PSTATE.GE on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to IRQ mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in IRQ mode.

SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to IRQ mode, and copied to PSTATE.E on executing an exception return operation in IRQ mode.

If the implementation does not support big-endian operation, SPSR_irq.E is **RES0**. If the implementation does not support little-endian operation, SPSR_irq.E is **RES1**. On executing an exception return operation in IRQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_irq.E is **RES0**, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_irq.E is **RES1**.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to IRQ mode, and copied to PSTATE.A on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to IRQ mode, and copied to PSTATE.I on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.
**F, bit [6]**

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to IRQ mode, and copied to PSTATE.F on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

**T, bit [5]**

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to IRQ mode, and copied to PSTATE.T on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

**M[4:0], bits [4:0]**

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to IRQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in IRQ mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined</td>
</tr>
<tr>
<td>0b11111</td>
<td>System</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_irq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in IRQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_irq**

Accesses to this register use the following encodings:

**MRS <Xt>, SPSR_irq**

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_irq;
elsif PSTATE.EL == EL3 then
    return SPSR_irq;
```

**MSR SPSR_irq, <Xt>**

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_irq;
elsif PSTATE.EL == EL3 then
    return SPSR_irq;
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_irq = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_irq = X[t];
SPSR_und, Saved Program Status Register (Undefined mode)

The SPSR_und characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to Undefined mode.

**Configuration**

AArch64 System register SPSR_und bits [31:0] are architecturally mapped to AArch32 System register SPSR_und[31:0].

If EL1 does not support execution in AArch32 state, this register is RES0 from EL2 and EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_und is a 64-bit register.

**Field descriptions**

The SPSR_und bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| N  | Z  | C  | V  | Q  | IT(1:0) | TSBS | P | AN | DI | IT | IL | GE | IT(7:2) | E | A | I | F | T | M[4:0] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:32]**

Reserved, RES0.

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Undefined mode, and copied to PSTATE.N on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Undefined mode, and copied to PSTATE.Z on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Undefined mode, and copied to PSTATE.C on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.
V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Undefined mode, and copied to PSTATE.V on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Undefined mode, and copied to PSTATE.Q on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

IT[1:0], bits [26:25]

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Undefined mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Undefined mode.

On executing an exception return operation in Undefined mode SPSR_und.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Undefined mode, and copied to PSTATE.SSBS on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Undefined mode, and copied to PSTATE.PAN on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
**DIT, bit [21]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Undefined mode, and copied to PSTATE.DIT on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Undefined mode, and copied to PSTATE.IL on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Undefined mode, and copied to PSTATE.GE on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Undefined mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in Undefined mode.

SPSR_und.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to Undefined mode, and copied to PSTATE.E on executing an exception return operation in Undefined mode.

If the implementation does not support big-endian operation, SPSR_und.E is **RES0**. If the implementation does not support little-endian operation, SPSR_und.E is **RES1**. On executing an exception return operation in Undefined mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_und.E is **RES0**, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_und.E is **RES1**.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Undefined mode, and copied to PSTATE.A on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Undefined mode, and copied to PSTATE.I on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.
**F, bit [6]**

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Undefined mode, and copied to PSTATE.F on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.

**T, bit [5]**

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Undefined mode, and copied to PSTATE.T on executing an exception return operation in Undefined mode.

This field resets to an architecturally **UNKNOWN** value.

**M[4:0], bits [4:0]**

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Undefined mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Undefined mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_und.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Undefined mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_und**

Accesses to this register use the following encodings:

**MRS <Xt>, SPSR_und**

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_und;
elsif PSTATE.EL == EL3 then
    return SPSR_und;
```

**MSR SPSR_und, <Xt>**

```plaintext
0b11 0b100 0b0100 0b0011 0b010
```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  SPSR_und = X[t];
elsif PSTATE.EL == EL3 then
  SPSR_und = X[t];
SSBS, Speculative Store Bypass Safe

The SSBS characteristics are:

**Purpose**

Allows access to the Speculative Store Bypass Safe bit.

**Configuration**

This register is present only when ARMv8.0-SSBS is implemented. Otherwise, direct accesses to SSBS are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

SSBS is a 64-bit register.

**Field descriptions**

The SSBS bit assignments are:

![SSBS Field Assignments]

**Bits [63:13]**

Reserved, RES0.

**SSBS, bit [12]**

Speculative Store Bypass Safe.

Prohibits speculative loads or stores which might practically allow a cache timing side channel.

A cache timing side channel might be exploited where a load or store uses an address that is derived from a register that is being loaded from memory using a load instruction speculatively read from a memory location. If PSTATE.SSBS is enabled, the address derived from the load instruction might be from earlier in the coherence order than the latest store to that memory location with the same virtual address.
<table>
<thead>
<tr>
<th>SSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hardware is not permitted to load or store speculatively, in a manner that could practically give rise to a cache timing side channel, using an address derived from a register value that has been loaded from memory using a load instruction (L) that speculatively reads an entry from earlier in the coherence order from that location being loaded from than the entry generated by the latest store (S) to that location using the same virtual address as L.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hardware is permitted to load or store speculatively, in a manner that could practically give rise to a cache timing side channel, using an address derived from a register value that has been loaded from memory using a load instruction (L) that speculatively reads an entry from earlier in the coherence order from that location being loaded from than the entry generated by the latest store (S) to that location using the same virtual address as L.</td>
</tr>
</tbody>
</table>

The value of this bit is set to the value in the SCTLR_ELx.DSSBS field on taking an exception to ELx.

This field resets to an IMPLEMENTATION DEFINED value.

**Bits [11:0]**

Reserved, RES0.

### Accessing the SSBS

For details on the operation of the MSR (immediate) accessor, see MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

**MRS <Xt>, SSBS**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```diff
if PSTATE.EL == EL0 then
  return Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL1 then
  return Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL2 then
  return Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL3 then
  return Zeros(51):PSTATE.SSBS:Zeros(12);
```

**MSR SSBS, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```diff
if PSTATE.EL == EL0 then
  PSTATE.SSBS = X[t]<12>;
elsif PSTATE.EL == EL1 then
  PSTATE.SSBS = X[t]<12>;
elsif PSTATE.EL == EL2 then
  PSTATE.SSBS = X[t]<12>;
elsif PSTATE.EL == EL3 then
  PSTATE.SSBS = X[t]<12>;
```
## MSR SSBS, #<imm>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TCO, Tag Check Override

The TCO characteristics are:

### Purpose

When ARMv8.5-MemTag is implemented, this register allows tag checks to be disabled globally.

### Configuration

This register is present only when ARMv8.5-MemTag is implemented. Otherwise, direct accesses to TCO are UNDEFINED.

### Attributes

TCO is a 64-bit register.

### Field descriptions

The TCO bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>TCO</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:26]**

Reserved, RES0.

**TCO, bit [25]**

Allows memory tag checks to be globally disabled.

<table>
<thead>
<tr>
<th>TCO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Loads and Stores are not affected by this control.</td>
</tr>
<tr>
<td>0b1</td>
<td>Loads and Stores are unchecked.</td>
</tr>
</tbody>
</table>

**Bits [24:0]**

Reserved, RES0.

### Accessing the TCO

For details on the operation of the MSR (immediate) accessor, see MSR (immediate).

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    return Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL1 then
    return Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL2 then
    return Zeros(38):PSTATE.TCO:Zeros(25);
elsif PSTATE.EL == EL3 then
    return Zeros(38):PSTATE.TCO:Zeros(25);

MSR TCO, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    PSTATE.TCO = X[t]<25>;
elsif PSTATE.EL == EL1 then
    PSTATE.TCO = X[t]<25>;
elsif PSTATE.EL == EL2 then
    PSTATE.TCO = X[t]<25>;
elsif PSTATE.EL == EL3 then
    PSTATE.TCO = X[t]<25>;

MSR TCO, #<imm>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TCR_EL1 characteristics are:

**Purpose**

The control register for stage 1 of the EL1&0 translation regime.

**Configuration**

AArch64 System register TCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TTBCR[31:0].

AArch64 System register TCR_EL1 bits [63:32] are architecturally mapped to AArch32 System register TTBCR2[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TCR_EL1 is a 64-bit register.

**Field descriptions**

The TCR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>TCMA1</td>
<td>Controls the generation of Unchecked accesses at EL1, and at EL0 if HCR_EL2.E2H,TGE!=1,1, when address[59:55] = 0b11111.</td>
</tr>
<tr>
<td>61</td>
<td>TCMA0</td>
<td>This control has no effect on the generation of Unchecked accesses at EL1 or EL0.</td>
</tr>
<tr>
<td>60</td>
<td>E0PD1</td>
<td>All accesses at EL1 and EL0 are Unchecked.</td>
</tr>
<tr>
<td>59</td>
<td>E0PD0</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>NFD1</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>NFD0</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>TBID1</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>TBID0</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>HWU161</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>HWU160</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>HWU061</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>HWU060</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>HWU059</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>HPD1</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>HPD0</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>HD</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>IPS</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>TG1</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>SH1</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>ORGN1</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>IRGN1</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>EPD1</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>T1SZ</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>TG0</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>T0SZ</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>TCMA1</td>
<td>Controls the generation of Unchecked accesses at EL1, and at EL0 if HCR_EL2.E2H,TGE!=1,1, when address[59:55] = 0b11111.</td>
</tr>
<tr>
<td>30</td>
<td>TCMA0</td>
<td>This control has no effect on the generation of Unchecked accesses at EL1 or EL0.</td>
</tr>
<tr>
<td>29</td>
<td>E0PD1</td>
<td>All accesses at EL1 and EL0 are Unchecked.</td>
</tr>
<tr>
<td>28</td>
<td>E0PD0</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>NFD1</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>NFD0</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>TBID1</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>TBID0</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>HWU161</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>HWU160</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>HWU061</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>HWU060</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>HWU059</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>HPD1</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>HPD0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>HD</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>AS</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>IPS</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>TG1</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>SH1</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ORGN1</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>IRGN1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>EPD1</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>T1SZ</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TG0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>T0SZ</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>RES0</td>
<td></td>
</tr>
</tbody>
</table>

Any of the bits in TCR_EL1, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

**Bits [63:59]**

Reserved, RES0.

**TCMA1, bit [58]**

**When ARMv8.5-MemTag is implemented:**

Controls the generation of Unchecked accesses at EL1, and at EL0 if HCR_EL2.E2H,TGE!=1,1, when address[59:55] = 0b11111.

<table>
<thead>
<tr>
<th>TCMA1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the generation of Unchecked accesses at EL1 or EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>All accesses at EL1 and EL0 are Unchecked.</td>
</tr>
</tbody>
</table>

**Note**

Software may change this control bit on a context switch.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**TCMA0, bit [57]**

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if HCR_EL2.E2H,TGE!=\{1,1\}, when address[59:55]=0b00000.

<table>
<thead>
<tr>
<th>TCMA0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the generation of Unchecked accesses at EL1 or EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>All accesses at EL1 and EL0 are Unchecked.</td>
</tr>
</tbody>
</table>

**Note**

Software may change this control bit on a context switch.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**E0PD1, bit [56]**

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR1_EL1.

<table>
<thead>
<tr>
<th>E0PD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unprivileged access to any address translated by TTBR1_EL1 will not generate a fault by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>Unprivileged access to any address translated by TTBR1_EL1 will generate a level 0 translation fault</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**E0PD0, bit [55]**

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR0_EL1.

<table>
<thead>
<tr>
<th>E0PD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unprivileged access to any address translated by TTBR0_EL1 will not generate a fault by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>Unprivileged access to any address translated by TTBR0_EL1 will generate a level 0 translation fault</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
NFD1, bit [54]

When SVE is implemented or TME is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR1_EL1.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR1_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for more information.

If TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

Defined values are:

<table>
<thead>
<tr>
<th>NFD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not disable stage 1 translation table walks using TTBR1_EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>A TLB miss on a virtual address that is translated using TTBR1_EL1 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]

When SVE is implemented or TME is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR0_EL1.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR0_EL1.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for more information.

If TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

Defined values are:
NFD0

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**TBID1, bit [52]**

*When ARMv8.3-PAuth is implemented:*

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>TBID1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TCR_EL1.TBI1 applies to Instruction and Data accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>TCR_EL1.TBI1 applies to Data accesses only.</td>
</tr>
</tbody>
</table>

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**TBID0, bit [51]**

*When ARMv8.3-PAuth is implemented:*

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>TBID0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TCR_EL1.TBI0 applies to Instruction and Data accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>TCR_EL1.TBI0 applies to Data accesses only.</td>
</tr>
</tbody>
</table>

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL1.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**HWU162, bit [50]**
When ARMv8.2-TTPBHA is implemented:

**Hardware Use.** Indicates **IMPLEMENTATION DEFINED** hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using \texttt{TTBR1_EL1}.

<table>
<thead>
<tr>
<th>HWU162</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using \texttt{TTBR1_EL1}, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using \texttt{TTBR1_EL1}, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose if the value of TCR_EL1.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**HWU161, bit [49]**

When ARMv8.2-TTPBHA is implemented:

**Hardware Use.** Indicates **IMPLEMENTATION DEFINED** hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using \texttt{TTBR1_EL1}.

<table>
<thead>
<tr>
<th>HWU161</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using \texttt{TTBR1_EL1}, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using \texttt{TTBR1_EL1}, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose if the value of TCR_EL1.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**HWU160, bit [48]**

When ARMv8.2-TTPBHA is implemented:

**Hardware Use.** Indicates **IMPLEMENTATION DEFINED** hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using \texttt{TTBR1_EL1}.

<table>
<thead>
<tr>
<th>HWU160</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using \texttt{TTBR1_EL1}, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using \texttt{TTBR1_EL1}, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose if the value of TCR_EL1.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HWU159, bit [47]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates **IMPLEMENTATION DEFINED** hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using **TTBR1_EL1**.

<table>
<thead>
<tr>
<th>HWU159</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using <strong>TTBR1_EL1</strong>, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using <strong>TTBR1_EL1</strong>, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose if the value of TCR_EL1.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HWU062, bit [46]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates **IMPLEMENTATION DEFINED** hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using **TTBR0_EL1**.

<table>
<thead>
<tr>
<th>HWU062</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using <strong>TTBR0_EL1</strong>, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using <strong>TTBR0_EL1</strong>, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose if the value of TCR_EL1.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HWU061, bit [45]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates **IMPLEMENTATION DEFINED** hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using **TTBR0_EL1**.
<table>
<thead>
<tr>
<th>HWU061</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using \texttt{TTBR0_EL1}, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an \texttt{IMPLEMENTATION DEFINED} purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using \texttt{TTBR0_EL1}, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an \texttt{IMPLEMENTATION DEFINED} purpose if the value of TCR_EL1.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR\_EL1.HPD0 is 0.

This field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, RES0.

**HWU060, bit [44]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates \texttt{IMPLEMENTATION DEFINED} hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using \texttt{TTBR0\_EL1}.

<table>
<thead>
<tr>
<th>HWU060</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using \texttt{TTBR0_EL1}, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an \texttt{IMPLEMENTATION DEFINED} purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using \texttt{TTBR0_EL1}, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an \texttt{IMPLEMENTATION DEFINED} purpose if the value of TCR_EL1.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR\_EL1.HPD0 is 0.

This field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, RES0.

**HWU059, bit [43]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates \texttt{IMPLEMENTATION DEFINED} hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using \texttt{TTBR0\_EL1}.

<table>
<thead>
<tr>
<th>HWU059</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using \texttt{TTBR0_EL1}, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an \texttt{IMPLEMENTATION DEFINED} purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using \texttt{TTBR0_EL1}, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an \texttt{IMPLEMENTATION DEFINED} purpose if the value of TCR_EL1.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR\_EL1.HPD0 is 0.

This field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, RES0.
HPD1, bit [42]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR1_EL1.

<table>
<thead>
<tr>
<th>HPD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL1.

<table>
<thead>
<tr>
<th>HPD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL0 and EL1.

<table>
<thead>
<tr>
<th>HD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 hardware management of dirty state disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL0 and EL1.
HA

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**TBI1, bit [38]**

Top Byte ignored - indicates whether the top byte of an address is used for address match for the **TTBR1_EL1** region, or ignored and used for tagged addresses. Defined values are:

<table>
<thead>
<tr>
<th>TBI1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Top Byte used in the address calculation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Top Byte ignored in the address calculation.</td>
</tr>
</tbody>
</table>

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by **TTBR1_EL1**. It has an effect whether the EL1&0 translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL1.TBID1 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally **UNKNOWN** value.

**TBI0, bit [37]**

Top Byte ignored - indicates whether the top byte of an address is used for address match for the **TTBR0_EL1** region, or ignored and used for tagged addresses. Defined values are:

<table>
<thead>
<tr>
<th>TBI0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Top Byte used in the address calculation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Top Byte ignored in the address calculation.</td>
</tr>
</tbody>
</table>

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by **TTBR0_EL1**. It has an effect whether the EL1&0 translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL1.TBID0 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally **UNKNOWN** value.

**AS, bit [36]**

ASID Size. Defined values are:
### AS

<table>
<thead>
<tr>
<th>AS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>8 bit - the upper 8 bits of TTBR0_EL1 and TTBR1_EL1 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.</td>
</tr>
<tr>
<td>0b1</td>
<td>16 bit - the upper 16 bits of TTBR0_EL1 and TTBR1_EL1 are used for allocation and matching in the TLB.</td>
</tr>
</tbody>
</table>

If the implementation has only 8 bits of ASID, this field is RES0.

This field resets to an architecturally UNKNOWN value.

### Bit [35]

Reserved, RES0.

### IPS, bits [34:32]

Intermediate Physical Address Size.

<table>
<thead>
<tr>
<th>IPS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>32 bits, 4GB.</td>
</tr>
<tr>
<td>0b001</td>
<td>36 bits, 64GB.</td>
</tr>
<tr>
<td>0b010</td>
<td>40 bits, 1TB.</td>
</tr>
<tr>
<td>0b011</td>
<td>42 bits, 4TB.</td>
</tr>
<tr>
<td>0b100</td>
<td>44 bits, 16TB.</td>
</tr>
<tr>
<td>0b101</td>
<td>48 bits, 256TB.</td>
</tr>
<tr>
<td>0b110</td>
<td>52 bits, 4PB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL1 are 0b0000.

This field resets to an architecturally UNKNOWN value.

### TG1, bits [31:30]

Granule size for the TTBR1 EL1.

<table>
<thead>
<tr>
<th>TG1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>16KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b11</td>
<td>64KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

### SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1_EL1.
Meaning

<table>
<thead>
<tr>
<th>SH1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**ORGN1, bits [27:26]**

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

<table>
<thead>
<tr>
<th>ORGN1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**IRGN1, bits [25:24]**

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL1.

<table>
<thead>
<tr>
<th>IRGN1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**EPD1, bit [23]**

Translation table walk disable for translations using TTBR1_EL1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1_EL1. The encoding of this bit is:

<table>
<thead>
<tr>
<th>EPD1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Perform translation table walks using TTBR1_EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>A TLB miss on an address that is translated using TTBR1_EL1 generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**A1, bit [22]**

Selects whether TTBR0_EL1 or TTBR1_EL1 defines the ASID. The encoding of this bit is:

<table>
<thead>
<tr>
<th>A1</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TTBR0_EL1 ASID defines the ASID.</td>
</tr>
<tr>
<td>0b1</td>
<td>TTBR1_EL1 ASID defines the ASID.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
**T1SZ, bits [21:16]**

The size offset of the memory region addressed by TTBR1_EL1. The region size is $2^{64-T1SZ}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

**TG0, bits [15:14]**

Granule size for the TTBR0_EL1.

<table>
<thead>
<tr>
<th>TG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB</td>
</tr>
<tr>
<td>0b01</td>
<td>64KB</td>
</tr>
<tr>
<td>0b10</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

**SH0, bits [13:12]**

Shareability attribute for memory associated with translation table walks using TTBR0_EL1.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in ‘Reserved values in AArch64 System registers and translation table entries’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**ORGN0, bits [11:10]**

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL1.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**IRGN0, bits [9:8]**

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL1.
IRGN0 | Meaning
---|---
0b00 | Normal memory, Inner Non-cacheable.
0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.
0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally **UNKNOWN** value.

**EPD0, bit [7]**

Translation table walk disable for translations using **TTBR0_EL1**. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using **TTBR0_EL1**. The encoding of this bit is:

<table>
<thead>
<tr>
<th>EPD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Perform translation table walks using <strong>TTBR0_EL1</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>A TLB miss on an address that is translated using <strong>TTBR0_EL1</strong> generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bit [6]**

Reserved, **RES0**.

**T0SZ, bits [5:0]**

The size offset of the memory region addressed by **TTBR0_EL1**. The region size is $2^{(64-\text{T0SZ})}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the TCR_EL1**

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL3 using the mnemonic **TCR_EL1** or **TCR_EL12** are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, TCR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return NVMem[0x120];
  end if
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return TCR_EL2;
  else
    return TCR_EL1;
  end if
elsif PSTATE.EL == EL3 then
  return TCR_EL1;
end if

MSR TCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TCR_EL1 = X[t];
  end if
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    TCR_EL2 = X[t];
  else
    TCR_EL1 = X[t];
  end if
elsif PSTATE.EL == EL3 then
  TCR_EL1 = X[t];
end if

MRS <Xt>, TCR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x120];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return TCR_EL1;
    else
        UNDEFINED;

MSR TCR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x120] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TCR_EL1 = X[t];
    else
        UNDEFINED;
The TCR_EL2 characteristics are:

**Purpose**

The control register for stage 1 of the EL2, or EL2&0, translation regime:

- When the Effective value of HCR_EL2.E2H is 0, this register controls stage 1 of the EL2 translation regime, that supports a single VA range, translated using TTBR0_EL2.
- When the value of HCR_EL2.E2H is 1, this register controls stage 1 of the EL2&0 translation regime, that supports both:
  - A lower VA range, translated using TTBR0_EL2.
  - A higher VA range, translated using TTBR1_EL2.

**Configuration**

AArch64 System register TCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTC[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TCR_EL2 is a 64-bit register.

**Field descriptions**

The TCR_EL2 bit assignments are:

**When HCR_EL2.E2H == 0:**

Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

**Bits [63:32]**

Reserved, RES0.

**Bit [31]**

Reserved, RES1.

**TCMA, bit [30]**

*When ARMv8.5-MemTag is implemented:*

Controls the generation of Unchecked accesses at EL2 when address [59:56] = 0b0000.
### TCMA

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>This control has no effect on the generation of Unchecked accesses.</td>
</tr>
<tr>
<td>1</td>
<td>All accesses are Unchecked.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### TBID, bit [29]

**When ARMv8.3-PAuth is implemented:**

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for AArch64-A architecture profile.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TCR_EL2.TBI applies to Instruction and Data accesses.</td>
</tr>
<tr>
<td>1</td>
<td>TCR_EL2.TBI applies to Data accesses only.</td>
</tr>
</tbody>
</table>

This affects addresses where the address would be translated by tables pointed to by **TTBR0_EL2**.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### HWU62, bit [28]

**When ARMv8.2-TTPHA is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>1</td>
<td>Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.
HWU61, bit [27]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU61</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU60, bit [26]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU60</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU59, bit [25]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU59</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

HPD, bit [24]

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL2.

<table>
<thead>
<tr>
<th>HPD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions disabled.</td>
</tr>
</tbody>
</table>

Note

In this case bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

<table>
<thead>
<tr>
<th>HD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 hardware management of dirty state disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL2.

<table>
<thead>
<tr>
<th>HA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 Access flag update disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 Access flag update enabled.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**TBI, bit [20]**

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the **TTBR0_EL2** region, or ignored and used for tagged addresses.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>TBI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Top Byte used in the address calculation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Top Byte ignored in the address calculation.</td>
</tr>
</tbody>
</table>

This affects addresses generated in EL2 using AArch64 where the address would be translated by tables pointed to by **TTBR0_EL2**. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL2.TBID is 1, then this field only applies to Data accesses.

If the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2.
- An exception taken to EL2.
- An exception return to EL2.

This field resets to an architecturally **UNKNOWN** value.

**Bit [19]**

Reserved, **RES0**.

**PS, bits [18:16]**

Physical Address Size.

<table>
<thead>
<tr>
<th>PS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>32 bits, 4GB.</td>
</tr>
<tr>
<td>0b001</td>
<td>36 bits, 64GB.</td>
</tr>
<tr>
<td>0b010</td>
<td>40 bits, 1TB.</td>
</tr>
<tr>
<td>0b011</td>
<td>42 bits, 4TB.</td>
</tr>
<tr>
<td>0b100</td>
<td>44 bits, 16TB.</td>
</tr>
<tr>
<td>0b101</td>
<td>48 bits, 256TB.</td>
</tr>
<tr>
<td>0b110</td>
<td>52 bits, 4PB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

The reserved values behave in the same way as the **0b101** or **0b110** encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value **0b110** is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not **0b110** or a value treated as **0b110**, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are **0b0000**.

This field resets to an architecturally **UNKNOWN** value.

**TG0, bits [15:14]**

Granule size for the **TTBR0_EL2**.
### TCR_EL2, Translation Control Register (EL2)

<table>
<thead>
<tr>
<th>TG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b01</td>
<td>64KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>16KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally **UNKNOWN** value.

**SH0, bits [13:12]**

Shareability attribute for memory associated with translation table walks using **TTBR0_EL2**.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONstrained UNPredictable, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**ORGN0, bits [11:10]**

Outer cacheability attribute for memory associated with translation table walks using **TTBR0_EL2**.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**IRGN0, bits [9:8]**

Inner cacheability attribute for memory associated with translation table walks using **TTBR0_EL2**.

<table>
<thead>
<tr>
<th>IRGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [7:6]**

Reserved, **RES0**.
**T0SZ, bits [5:0]**

The size offset of the memory region addressed by $TTBR0_{EL2}$. The region size is $2^{64-T0SZ}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally **UNKNOWN** value.

**When ARMv8.1-VHE is implemented and HCR_EL2.E2H == 1:**

Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

**Bits [63:59]**

Reserved, RES0.

**TCMA1, bit [58]**

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b111111.

<table>
<thead>
<tr>
<th>TCMA1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the generation of Unchecked accesses at EL2 or EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>All accesses are Unchecked.</td>
</tr>
</tbody>
</table>

**Note**

Software may change this control bit on a context switch.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**TCMA0, bit [57]**

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b000000.

<table>
<thead>
<tr>
<th>TCMA0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the generation of Unchecked accesses at EL2 or EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>All accesses are Unchecked.</td>
</tr>
</tbody>
</table>

**Note**

Software may change this control bit on a context switch.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**E0PD1, bit [56]**

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR1_EL2.

<table>
<thead>
<tr>
<th>E0PD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unprivileged access to any address translated by TTBR1_EL2 will not generate a fault by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>Unprivileged access to any address translated by TTBR1_EL2 will generate a level 0 translation fault</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**E0PD0, bit [55]**

When ARMv8.5-E0PD is implemented:

Faulting control for EL0 access to any address translated by TTBR0_EL2.

<table>
<thead>
<tr>
<th>E0PD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unprivileged access to any address translated by TTBR0_EL2 will not generate a fault by this mechanism.</td>
</tr>
<tr>
<td>0b1</td>
<td>Unprivileged access to any address translated by TTBR0_EL2 will generate a level 0 translation fault</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**NFD1, bit [54]**

When SVE is implemented or TME is implemented:

Non-fault translation table walk disable for stage 1 translations using TTBR1_EL2.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR1_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

See 'The Scalable Vector Extension (SVE)', in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for more information.

If TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

Defined values are:
### NFD1

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### NFD0, bit [53]

**When SVE is implemented or TME is implemented:**

Non-fault translation table walk disable for stage 1 translations using TTBR0_EL2.

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault access from EL0 for a virtual address that is translated using TTBR0_EL2.

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.


If TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

Defined values are:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### TBID1, bit [52]

**When ARMv8.3-PAuth is implemented:**

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see ‘Address tagging in AArch64 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
<table>
<thead>
<tr>
<th>TBID1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TCR_EL2.TBI1 applies to Instruction and Data accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>TCR_EL2.TBI1 applies to Data accesses only.</td>
</tr>
</tbody>
</table>

This affects addresses where the address would be translated by tables pointed to by TTBR1_EL2.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**TBID0, bit [51]**

*When ARMv8.3-PAuth is implemented:*

Controls the use of the top byte of instruction addresses for address matching.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>TBID0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TCR_EL2.TBI0 applies to Instruction and Data accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>TCR_EL2.TBI0 applies to Data accesses only.</td>
</tr>
</tbody>
</table>

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL2.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**HWU162, bit [50]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

<table>
<thead>
<tr>
<th>HWU162</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR1_EL2, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**HWU161, bit [49]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.
| HWU161 Meaning | 0b0 | For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| | 0b1 | For translations using TTBR1_EL2, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU160, bit [48]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

| HWU160 Meaning | 0b0 | For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| | 0b1 | For translations using TTBR1_EL2, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU159, bit [47]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1_EL2.

| HWU159 Meaning | 0b0 | For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| | 0b1 | For translations using TTBR1_EL2, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
HWU062, bit [46]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

<table>
<thead>
<tr>
<th>HWU062</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR0_EL1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU061, bit [45]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

<table>
<thead>
<tr>
<th>HWU061</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR0_EL1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU060, bit [44]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0_EL1.

<table>
<thead>
<tr>
<th>HWU060</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR0_EL1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HWU059, bit [43]**

When ARMv8.2-TTPBHA is implemented:

**Hardware Use.** Indicates **IMPLEMENTATION DEFINED** hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using **TTBR0_EL1**.

<table>
<thead>
<tr>
<th>HWU059</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using <strong>TTBR0_EL1</strong>, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using <strong>TTBR0_EL1</strong>, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HPD1, bit [42]**

When ARMv8.1-HPD is implemented:

**Hierarchical Permission Disables.** This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by **TTBR1_EL2**.

<table>
<thead>
<tr>
<th>HPD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HPD0, bit [41]**

When ARMv8.1-HPD is implemented:

**Hierarchical Permission Disables.** This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by **TTBR0_EL2**.

<table>
<thead>
<tr>
<th>HPD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**HD, bit [40]**

*When ARMv8.1-TTHM is implemented:*

Hardware management of dirty state in stage 1 translations from EL2.

<table>
<thead>
<tr>
<th>HD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 hardware management of dirty state disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**HA, bit [39]**

*When ARMv8.1-TTHM is implemented:*

Hardware Access flag update in stage 1 translations from EL2.

<table>
<thead>
<tr>
<th>HA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 Access flag update disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 Access flag update enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**TBI1, bit [38]**

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR1_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>TBI1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Top Byte used in the address calculation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Top Byte ignored in the address calculation.</td>
</tr>
</tbody>
</table>

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR1_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL2.TBID1 is 1, then this field only applies to Data accesses.

If the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally **UNKNOWN** value.
**TBI0, bit [37]**

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL2 region, or ignored and used for tagged addresses.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

<table>
<thead>
<tr>
<th>TBI0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Top Byte used in the address calculation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Top Byte ignored in the address calculation.</td>
</tr>
</tbody>
</table>

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL2. It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL2.TBID0 is 1, then this field only applies to Data accesses.

If the value of TBI0 is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

This field resets to an architecturally **UNKNOWN** value.

**AS, bit [36]**

ASID Size. Defined values are:

<table>
<thead>
<tr>
<th>AS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>8 bit - the upper 8 bits of TTBR0_EL2 and TTBR1_EL2 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB.</td>
</tr>
<tr>
<td>0b1</td>
<td>16 bit - the upper 16 bits of TTBR0_EL2 and TTBR1_EL2 are used for allocation and matching in the TLB.</td>
</tr>
</tbody>
</table>

If the implementation has only 8 bits of ASID, this field is **RES0**.

This field resets to an architecturally **UNKNOWN** value.

**Bit [35]**

Reserved, **RES0**.

**IPS, bits [34:32]**

Intermediate Physical Address Size.

<table>
<thead>
<tr>
<th>IPS</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>32 bits, 4GB.</td>
<td></td>
</tr>
<tr>
<td>0b001</td>
<td>36 bits, 64GB.</td>
<td></td>
</tr>
<tr>
<td>0b010</td>
<td>40 bits, 1TB.</td>
<td></td>
</tr>
<tr>
<td>0b011</td>
<td>42 bits, 4TB.</td>
<td></td>
</tr>
<tr>
<td>0b100</td>
<td>44 bits, 16TB.</td>
<td></td>
</tr>
<tr>
<td>0b101</td>
<td>48 bits, 256TB.</td>
<td></td>
</tr>
<tr>
<td>0b110</td>
<td>52 bits, 4PB.</td>
<td>When ARMv8.2-LPA is implemented</td>
</tr>
</tbody>
</table>

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.
In an implementation that supports 52-bit PAs, if the value of this field is not \(0b110\) or a value treated as \(0b110\), then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are \(0b0000\).

This field resets to an architecturally **UNKNOWN** value.

**TG1, bits [31:30]**

Granule size for the TTBR1_EL2.

<table>
<thead>
<tr>
<th>TG1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>16KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b11</td>
<td>64KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally **UNKNOWN** value.

**SH1, bits [29:28]**

Shareability attribute for memory associated with translation table walks using TTBR1_EL2. Defined values are:

<table>
<thead>
<tr>
<th>SH1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**ORGN1, bits [27:26]**

Outer cacheability attribute for memory associated with translation table walks using TTBR1_EL2.

<table>
<thead>
<tr>
<th>ORGN1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**IRGN1, bits [25:24]**

Inner cacheability attribute for memory associated with translation table walks using TTBR1_EL2.
### IRGN1

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### EPD1, bit [23]

Translation table walk disable for translations using `TTBR1_EL2`. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using `TTBR1_EL2`. The encoding of this bit is:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Perform translation table walks using <code>TTBR1_EL2</code>.</td>
</tr>
<tr>
<td>0b1</td>
<td>A TLB miss on an address that is translated using <code>TTBR1_EL2</code> generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### A1, bit [22]

Selects whether `TTBR0_EL2` or `TTBR1_EL2` defines the ASID. The encoding of this bit is:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><code>TTBR0_EL2</code>.ASID defines the ASID.</td>
</tr>
<tr>
<td>0b1</td>
<td><code>TTBR1_EL2</code>.ASID defines the ASID.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### T1SZ, bits [21:16]

The size offset of the memory region addressed by `TTBR1_EL2`. The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally **UNKNOWN** value.

### TG0, bits [15:14]

Granule size for the `TTBR0_EL2`.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b01</td>
<td>64KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>16KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an **IMPLEMENTATION DEFINED** choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is **IMPLEMENTATION DEFINED** whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally **UNKNOWN** value.
SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using TTBR0_EL2.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Read-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL2.

<table>
<thead>
<tr>
<th>IRGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Read-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using TTBR0_EL2. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR0_EL2. The encoding of this bit is:

<table>
<thead>
<tr>
<th>EPD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Perform translation table walks using TTBR0_EL2</td>
</tr>
<tr>
<td>0b1</td>
<td>A TLB miss on an address that is translated using TTBR0_EL2 generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.
T0SZ, bits [5:0]

The size offset of the memory region addressed by TTBR0_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

This field resets to an architecturally UNKNOWN value.

**Accessing the TCR_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic TCR_EL2 or TCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, TCR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   return TCR_EL2;
else if PSTATE.EL == EL3 then
   return TCR_EL2;

MSR TCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   TCR_EL2 = X[t];
else if PSTATE.EL == EL3 then
   TCR_EL2 = X[t];

MRS <Xt>, TCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TCR_EL1;
  end
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return TCR_EL2;
  else
    return TCR_EL1;
  end
elsif PSTATE.EL == EL3 then
  return TCR_EL1;
end

MSR TCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TCR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TCR_EL1 = X[t];
  end
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    TCR_EL2 = X[t];
  else
    TCR_EL1 = X[t];
  end
elsif PSTATE.EL == EL3 then
  TCR_EL1 = X[t];
TCR_EL3, Translated Control Register (EL3)

The TCR_EL3 characteristics are:

**Purpose**

The control register for stage 1 of the EL3 translation regime.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to TCR_EL3 are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

TCR_EL3 is a 64-bit register.

**Field descriptions**

The TCR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES1</td>
</tr>
<tr>
<td>61</td>
<td>TCMA</td>
</tr>
<tr>
<td>60</td>
<td>TBID</td>
</tr>
<tr>
<td>59</td>
<td>HWU62</td>
</tr>
<tr>
<td>58</td>
<td>HWU61</td>
</tr>
<tr>
<td>57</td>
<td>HWU60</td>
</tr>
<tr>
<td>56</td>
<td>HWU59</td>
</tr>
<tr>
<td>55</td>
<td>HPD</td>
</tr>
<tr>
<td>54</td>
<td>RES2</td>
</tr>
<tr>
<td>53</td>
<td>HD</td>
</tr>
<tr>
<td>52</td>
<td>HA</td>
</tr>
<tr>
<td>51</td>
<td>TBI</td>
</tr>
<tr>
<td>50</td>
<td>RES1</td>
</tr>
<tr>
<td>49</td>
<td>PS</td>
</tr>
<tr>
<td>48</td>
<td>TG0</td>
</tr>
<tr>
<td>47</td>
<td>ORGN0</td>
</tr>
<tr>
<td>46</td>
<td>IRGN0</td>
</tr>
<tr>
<td>45</td>
<td>RES0</td>
</tr>
<tr>
<td>44</td>
<td>T0SZ</td>
</tr>
<tr>
<td>43</td>
<td>RES0</td>
</tr>
<tr>
<td>42</td>
<td>RES0</td>
</tr>
<tr>
<td>41</td>
<td>RES0</td>
</tr>
<tr>
<td>40</td>
<td>RES0</td>
</tr>
<tr>
<td>39</td>
<td>RES0</td>
</tr>
<tr>
<td>38</td>
<td>RES0</td>
</tr>
<tr>
<td>37</td>
<td>RES0</td>
</tr>
<tr>
<td>36</td>
<td>RES0</td>
</tr>
<tr>
<td>35</td>
<td>RES0</td>
</tr>
<tr>
<td>34</td>
<td>RES0</td>
</tr>
<tr>
<td>33</td>
<td>RES0</td>
</tr>
<tr>
<td>32</td>
<td>RES0</td>
</tr>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>RES0</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>RES0</td>
</tr>
<tr>
<td>25</td>
<td>RES0</td>
</tr>
<tr>
<td>24</td>
<td>RES0</td>
</tr>
<tr>
<td>23</td>
<td>RES0</td>
</tr>
<tr>
<td>22</td>
<td>RES0</td>
</tr>
<tr>
<td>21</td>
<td>RES0</td>
</tr>
<tr>
<td>20</td>
<td>RES0</td>
</tr>
<tr>
<td>19</td>
<td>RES0</td>
</tr>
<tr>
<td>18</td>
<td>RES0</td>
</tr>
<tr>
<td>17</td>
<td>RES0</td>
</tr>
<tr>
<td>16</td>
<td>RES0</td>
</tr>
<tr>
<td>15</td>
<td>RES0</td>
</tr>
<tr>
<td>14</td>
<td>RES0</td>
</tr>
<tr>
<td>13</td>
<td>RES0</td>
</tr>
<tr>
<td>12</td>
<td>RES0</td>
</tr>
<tr>
<td>11</td>
<td>RES0</td>
</tr>
<tr>
<td>10</td>
<td>RES0</td>
</tr>
<tr>
<td>9</td>
<td>RES0</td>
</tr>
<tr>
<td>8</td>
<td>RES0</td>
</tr>
<tr>
<td>7</td>
<td>RES0</td>
</tr>
<tr>
<td>6</td>
<td>RES0</td>
</tr>
<tr>
<td>5</td>
<td>RES0</td>
</tr>
<tr>
<td>4</td>
<td>RES0</td>
</tr>
<tr>
<td>3</td>
<td>RES0</td>
</tr>
<tr>
<td>2</td>
<td>RES0</td>
</tr>
<tr>
<td>1</td>
<td>RES0</td>
</tr>
<tr>
<td>0</td>
<td>RES0</td>
</tr>
</tbody>
</table>

Any of the bits in TCR_EL3 are permitted to be cached in a TLB.

**Bits [63:32]**

Reserved, RES0.

**Bit [31]**

Reserved, RES1.

**TCMA, bit [30]**

When ARMv8.5-MemTag is implemented:

Controls the generation of Unchecked accesses at EL3 when address [59:56] = 0b0000.

<table>
<thead>
<tr>
<th>TCMA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the generation of Unchecked accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>All accesses are Unchecked.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.
TBID, bit [29]

When ARMv8.3-PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

<table>
<thead>
<tr>
<th>TBID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TCR_EL3.TBI applies to Instruction and Data accesses.</td>
</tr>
<tr>
<td>0b1</td>
<td>TCR_EL3.TBI applies to Data accesses only.</td>
</tr>
</tbody>
</table>

This affects addresses where the address would be translated by tables pointed to by TTBR0_EL3.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU62</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU61, bit [27]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU61</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**HWU60, bit [26]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU60</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU59, bit [25]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU59</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HPD, bit [24]**

When ARMv8.1-HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by TTBR0_EL3.
<table>
<thead>
<tr>
<th>HPD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled.</td>
</tr>
</tbody>
</table>

**Note**

In this case bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bit [23]**

Reserved, RES1.

**HD, bit [22]**

When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 1 translations from EL3.

<table>
<thead>
<tr>
<th>HD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 hardware management of dirty state disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**HA, bit [21]**

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in stage 1 translations from EL3.

<table>
<thead>
<tr>
<th>HA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 1 Access flag update disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 1 Access flag update enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**TBI, bit [20]**

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the TTBR0_EL3 region, or ignored and used for tagged addresses.
TBI

<table>
<thead>
<tr>
<th>TBI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Top Byte used in the address calculation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Top Byte ignored in the address calculation.</td>
</tr>
</tbody>
</table>

This affects addresses generated in EL3 using AArch64 where the address would be translated by tables pointed to by TTBR0_EL3. It has an effect whether the EL3 translation regime is enabled or not.

If ARMv8.3-PAuth is implemented and TCR_EL3.TBID is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL3.
- A exception taken to EL3.
- An exception return to EL3.

For more information, see 'Address tagging in AArch64 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Note**

This control determines the scope of address tagging. It never causes an exception to be generated.

This field resets to an architecturally **UNKNOWN** value.

**Bit [19]**

Reserved, RES0.

**PS, bits [18:16]**

Physical Address Size.

<table>
<thead>
<tr>
<th>PS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>32 bits, 4GB.</td>
</tr>
<tr>
<td>0b01</td>
<td>36 bits, 64GB.</td>
</tr>
<tr>
<td>0b10</td>
<td>40 bits, 1TB.</td>
</tr>
<tr>
<td>0b11</td>
<td>42 bits, 4TB.</td>
</tr>
<tr>
<td>0b10</td>
<td>44 bits, 16TB.</td>
</tr>
<tr>
<td>0b10</td>
<td>48 bits, 256TB.</td>
</tr>
<tr>
<td>0b11</td>
<td>52 bits, 4PB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL3 are 0b0000.

This field resets to an architecturally **UNKNOWN** value.

**TG0, bits [15:14]**

Granule size for the TTBR0_EL3.

<table>
<thead>
<tr>
<th>TG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b01</td>
<td>64KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>16KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.
If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

**SH0, bits [13:12]**

Shareability attribute for memory associated with translation table walks using TTBR0_EL3.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

This field resets to an architecturally UNKNOWN value.

**ORGN0, bits [11:10]**

Outer cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**IRGN0, bits [9:8]**

Inner cacheability attribute for memory associated with translation table walks using TTBR0_EL3.

<table>
<thead>
<tr>
<th>IRGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Bits [7:6]**

Reserved, RES0.

**T0SZ, bits [5:0]**

The size offset of the memory region addressed by TTBR0_EL3. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.
This field resets to an architecturally **UNKNOWN** value.

## Accessing the TCR_EL3

Accesses to this register use the following encodings:

**MRS <Xt>, TCR_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if `PSTATE.EL == EL0` then
    UNDEFINED;
elsif `PSTATE.EL == EL1` then
    UNDEFINED;
elsif `PSTATE.EL == EL2` then
    UNDEFINED;
elsif `PSTATE.EL == EL3` then
    return TCR_EL3;

**MSR TCR_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if `PSTATE.EL == EL0` then
    UNDEFINED;
elsif `PSTATE.EL == EL1` then
    UNDEFINED;
elsif `PSTATE.EL == EL2` then
    UNDEFINED;
elsif `PSTATE.EL == EL3` then
    `TCR_EL3 = X[t];`
TFSR_EL1, Tag Fail Status Register (EL1)

The TFSR_EL1 characteristics are:

**Purpose**

Holds accumulated Tag Check Fails occurring in EL1 which are not taken precisely.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to TFSR_EL1 are **UNDEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

TFSR_EL1 is a 64-bit register.

**Field descriptions**

The TFSR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit&lt;55&gt;=0b1 occurs. This field resets to an architecturally <strong>UNKNOWN</strong> value.</td>
</tr>
<tr>
<td>0</td>
<td>Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit&lt;55&gt;=0b0 occurs. This field resets to an architecturally <strong>UNKNOWN</strong> value.</td>
</tr>
</tbody>
</table>

**Accessing the TFSR_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, TFSR_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b00</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x190];
    else
        return TFSR_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TFSR_EL2;
    else
        return TFSR_EL1;
    end
elsif PSTATE.EL == EL3 then
    return TFSR_EL1;
end

MSR TFSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x190] = X[t];
    else
        TFSR_EL1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TFSR_EL2 = X[t];
    else
        TFSR_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t];
end

MRS <Xt>, TFSR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x190];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TFSR_EL1 = X[t];
    else
        UNDEFINED;
MRS TFSR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x190];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TFSR_EL1 = X[t];
    else
        UNDEFINED;
MRS <Xt>, TFSR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSR_EL1;
        endif
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TFSR_EL2;
    endif
elsif PSTATE.EL == EL3 then
    return TFSR_EL2;
endif

MSR TFSR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
        endif
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TFSR_EL2 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    TFSR_EL2 = X[t];
The TFSR_EL2 characteristics are:

**Purpose**

Holds accumulated Tag Check Fails occurring in EL2 which are not taken precisely.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to TFSR_EL2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TFSR_EL2 is a 64-bit register.

**Field descriptions**

The TFSR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0 |

**Bits [63:2]**

Reserved, RES0.

**TF1, bit [1]**

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==0b1 occurs.

When HCR_EL2.E2H==0b0, this field is RES0.

This field resets to an architecturally UNKNOWN value.

**TF0, bit [0]**

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==0b0 occurs.

This field resets to an architecturally UNKNOWN value.

**Accessing the TFSR_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic TFSR_EL2 or TFSR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, TFSR_EL2

| op0 | op1 | CRn | CRm | op2 |
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TFSR_EL1;
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TFSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  TFSR_EL2 = X[t];
MRS <Xt>, TFSR_EL1

MSR TFSR_EL2, <Xt>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TFSR_EL1 = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TFSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  TFSR_EL2 = X[t];
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x190];
  else
    return TFSR_EL1;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    return TFSR_EL2;
  else
    return TFSR_EL1;
  endif
elsif PSTATE.EL == EL3 then
  return TFSR_EL1;
endif

MSR TFSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x190] = X[t];
  else
    TFSR_EL1 = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HCR_EL2.E2H == '1' then
    TFSR_EL2 = X[t];
  else
    TFSR_EL1 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  TFSR_EL1 = X[t];
TFSR_EL3, Tag Fail Status Register (EL3)

The TFSR_EL3 characteristics are:

**Purpose**

Holds accumulated Tag Check Fails occurring in EL3 which are not taken precisely.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to TFSR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TFSR_EL3 is a 64-bit register.

**Field descriptions**

The TFSR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>TF0, bit [0]</td>
</tr>
</tbody>
</table>

**Bits [63:1]**

Reserved, RES0.

**TF0, bit [0]**

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>=0b0 occurs. This field resets to an architecturally UNKNOWN value.

**Accessing the TFSR_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, TFSR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
 UNDEFINED;
elsif PSTATE.EL == EL1 then
 UNDEFINED;
elsif PSTATE.EL == EL2 then
 UNDEFINED;
elsif PSTATE.EL == EL3 then
 return TFSR_EL3;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TFSR_EL3 = X[t];
TFSRE0_EL1, Tag Fail Status Register (EL0).

The TFSRE0_EL1 characteristics are:

**Purpose**

Holds accumulated Tag Check Fails occurring in EL0 which are not taken precisely.

**Configuration**

This register is present only when ARMv8.5-MemTag is implemented and ID_AA64PFR1_EL1.MTE != 0b0001. Otherwise, direct accesses to TFSRE0_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TFSRE0_EL1 is a 64-bit register.

**Field descriptions**

The TFSRE0_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:2]</th>
<th>RES0</th>
<th>RES0</th>
<th>TF1</th>
<th>TF0</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17</td>
<td>16 15 14 13 12 11 10 9 8 7 6 5 4 3 2</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:2]**

Reserved, RES0.

**TF1, bit [1]**

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==0b1 occurs. This field resets to an architecturally UNKNOWN value.

**TF0, bit [0]**

Tag Check Fail. Asynchronously set to 1 when a Tag Check fail using a virtual address with bit<55>==0b0 occurs. This field resets to an architecturally UNKNOWN value.

**Accessing the TFSRE0_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, TFSRE0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TFSRE0_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TFSRE0_EL1;
elsif PSTATE.EL == EL3 then
  return TFSRE0_EL1;

MSR TFSRE0_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ATA == '0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TFSRE0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.ATA == '0' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TFSRE0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  TFSRE0_EL1 = X[t];
TLBI ALLE1, TLB Invalidate All, EL1

The TLBI ALLE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
- If SCR_EL3.NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation only applies to the PE that executes this System instruction.

**Note**

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI ALLE1 is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE1 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE1 instruction**

Accesses to this instruction use the following encodings:

TLBI ALLE1{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE1();
elsif PSTATE.EL == EL3 then
    TLBI_ALLE1();
The TLBI ALLE1IS characteristics are:

### Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If \( \text{SCR}_{\text{EL3}}.NS \) is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
- If \( \text{SCR}_{\text{EL3}}.NS \) is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

### Note

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

### Configuration

### Attributes

TLBI ALLE1IS is a 64-bit System instruction.

### Field descriptions

TLBI ALLE1IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

### Executing the TLBI ALLE1IS instruction

Accesses to this instruction use the following encodings:

TLBI ALLE1IS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b100</td>
<td>0b11111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE1IS();
elsif PSTATE.EL == EL3 then
    TLBI_ALLE1IS();
TLBI ALLE1OS, TLB Invalidate All, EL1, Outer Shareable

The TLBI ALLE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If `SCR_EL3.NS` is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
- If `SCR_EL3.NS` is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ALLE1OS are UNDEFINED.

**Attributes**

TLBI ALLE1OS is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE1OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE1OS instruction**

Accesses to this instruction use the following encodings:

TLBI ALLE10S{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b100</td>
<td>0b11111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE1OS();
eelsif PSTATE.EL == EL3 then
    TLBI_ALLE1OS();
The TLBI ALLE2 characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If \( \text{SCR}_{\text{EL3}}.\text{NS} = 1 \) and the entry would be required to translate an address using the Non-secure EL2 or Non-secure EL2\&0 translation regime.
- If \( \text{SCR}_{\text{EL3}}.\text{NS} = 0 \) and the entry would be required to translate an address using the Secure EL2 or Secure EL2\&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI ALLE2 is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE2 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE2 instruction**

Accesses to this instruction use the following encodings:

```
TLBI ALLE2{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        TLBI_ALLE2();
    elsif PSTATE.EL == EL3 then
        if !EL2Enabled() then
            UNDEFINED;
        else
            TLBI_ALLE2();
    ```
The TLBI ALLE2IS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If \texttt{SCR\_EL3}.\texttt{NS} is 1 and the entry would be required to translate an address using the Non-secure EL2 or Non-secure EL2&0 translation regime.
- If \texttt{SCR\_EL3}.\texttt{NS} is 0 and the entry would be required to translate an address using the Secure EL2 or Secure EL2&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI ALLE2IS is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE2IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE2IS instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI ALLE2IS\{, <Xt>\}}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if \texttt{PSTATE.EL} == \texttt{EL0} then
  UNDEFINED;
elsif \texttt{PSTATE.EL} == \texttt{EL1} then
  if \texttt{EL2Enabled()} && \texttt{HCR\_EL2.NV} == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif \texttt{PSTATE.EL} == \texttt{EL2} then
  TLBI\_ALLE2IS();
elsif \texttt{PSTATE.EL} == \texttt{EL3} then
  if !\texttt{EL2Enabled()} then
    UNDEFINED;
  else
    TLBI\_ALLE2IS();
The TLBI ALLE2OS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If $SCR_{EL3}.NS$ is 1 and the entry would be required to translate an address using the Non-secure EL2 or Non-secure EL2&0 translation regime.
- If $SCR_{EL3}.NS$ is 0 and the entry would be required to translate an address using the Secure EL2 or Secure EL2&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ALLE2OS are **UNDEFINED**.

**Attributes**

TLBI ALLE2OS is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE2OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE2OS instruction**

Accesses to this instruction use the following encodings:

TLBI ALLE20S{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b11111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_ALLE2OS();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_ALLE2OS();

TLBI ALLE3, TLB Invalidate All, EL3

The TLBI ALLE3 characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI ALLE3 is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE3 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE3 instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI ALLE3\{, <Xt}>}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
</table>
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b000 | 0b1111

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_ALLE3();
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TLBI ALLE3IS, TLB Invalidate All, EL3, Inner Shareable

The TLBI ALLE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

Attributes

TLBI ALLE3IS is a 64-bit System instruction.

Field descriptions

TLBI ALLE3IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

Executing the TLBI ALLE3IS instruction

Accesses to this instruction use the following encodings:

TLBI ALLE3IS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_ALLE3IS();
TLBI ALLE3OS, TLB Invalidate All, EL3, Outer Shareable

The TLBI ALLE3OS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ALLE3OS are UNDEFINED.

**Attributes**

TLBI ALLE3OS is a 64-bit System instruction.

**Field descriptions**

TLBI ALLE3OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI ALLE3OS instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI ALLE3OS\{, <Xt>\}}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b11111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_ALLE3OS();
TLBI ASIDE1, TLB Invalidate by ASID, EL1

The TLBI ASIDE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID, and would be required to translate an address using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI ASIDE1 is a 64-bit System instruction.

**Field descriptions**

The TLBI ASIDE1 input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ASID | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**ASID, bits [63:48]**

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

**Bits [47:0]**

Reserved, RES0.

**Executing the TLBI ASIDE1 instruction**

Accesses to this instruction use the following encodings:
TLBI ASIDE1, TLB Invalidate by ASID, EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
      AArch64.SystemAccessTrap(EL2, 0x10);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIASIDE1 == '1' then
      TLBI_ASIDE1IS(X[t]);
   else
      TLBI_ASIDE1(X[t]);
   elsif PSTATE.EL == EL2 then
      TLBI_ASIDE1(X[t]);
   elsif PSTATE.EL == EL3 then
      TLBI_ASIDE1(X[t]);
   end if
else
   TLBI ASIDE1(X[t]);
end if
The TLBI ASIDE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3 NS:
  - If HCR_EL2 \{E2H, TGE\} is not \{1, 1\}, the entry would be used with the current VMID, and would be required to translate an address using the EL1&0 translation regime.
  - If HCR_EL2 \{E2H, TGE\} is \{1, 1\}, the entry would be required to translate an address using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI ASIDE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI ASIDE1IS input value bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ASID | RES0 |

**ASID, bits [63:48]**

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

**Bits [47:0]**

Reserved, RES0.
Executing the TLBI ASIDE1IS instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI ASIDE1IS, } <\text{Xt}>\}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIASIDE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_ASIDE1IS(X[t]);
  elsif PSTATE.EL == EL2 then
    TLBI_ASIDE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_ASIDE1IS(X[t]);

The TLBI ASIDE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not \{1, 1\}, the entry would be used with the current VMID, and would be required to translate an address using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is \{1, 1\}, the entry would be required to translate an address using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI ASIDE1OS are **UNDEFINED**.

**Attributes**

TLBI ASIDE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI ASIDE1OS input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ASID | RES0 |

**ASID, bits [63:48]**

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are `RES0`.
Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1OS instruction

Accesses to this instruction use the following encodings:

TLBI ASIDE1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
       HFGITR_EL2.TLBIASIDE1OS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI ASIDE1OS(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI ASIDE1OS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI ASIDE1OS(X[t]);
endif
The TLBI IPAS2E1 characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - `SCR_EL3.NS==1` and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - `SCR_EL3.NS==0` and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

**Attributes**

TLBI IPAS2E1 is a 64-bit System instruction.

**Field descriptions**

The TLBI IPAS2E1 input value bit assignments are:

<p>| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|</p>
<table>
<thead>
<tr>
<th>NS</th>
<th>RES</th>
<th>TTL</th>
<th>RES</th>
<th>IPA[51:48]</th>
<th>IPA[47:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
</tbody>
</table>

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is `RES0`, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is `RES0`. 
Otherwise:

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

**Bits [43:40]**

Reserved, RES0.

**IPA[51:48], bits [39:36]**

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

Otherwise:

Reserved, RES0.
IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits
of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form
the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1 instruction

Accesses to this instruction use the following encodings:

TLBI IPAS2E1{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2E1(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2E1(X[t]);

The TLBI IPAS2E1IS characteristics are:

### Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - $\text{SCR}_{\text{EL3}}.\text{NS}==1$ and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - $\text{SCR}_{\text{EL3}}.\text{NS}==0$ and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

### Configuration

#### Attributes

TLBI IPAS2E1IS is a 64-bit System instruction.

#### Field descriptions

The TLBI IPAS2E1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS[63]</td>
<td></td>
<td>Not Secure. Specifies the IPA space.</td>
</tr>
<tr>
<td>NS[62]</td>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>NS[62]</td>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
<tr>
<td>TTL[49:48]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPA[51:48]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IPA[47:12]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.
Otherwise:

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TTL, bits [47:44]**

**When ARMv8.4-TTL is implemented:**

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
</tbody>
</table>
| 0b01xx  | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:  
          | 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.  
          | 0b01 : Level 1.  
          | 0b10 : Level 2.  
          | 0b11 : Level 3.  |
| 0b10xx  | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:  
          | 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.  
          | 0b01 : Reserved. Treat as if TTL<3:2> is 0b00.  
          | 0b10 : Level 2.  
          | 0b11 : Level 3.  |
| 0b11xx  | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as:  
          | 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.  
          | 0b01 : Level 1.  
          | 0b10 : Level 2.  
          | 0b11 : Level 3.  |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**Bits [43:40]**

Reserved, RES0.

**IPA[51:48], bits [39:36]**

**When ARMv8.2-LPA is implemented:**

Extension to IPA[47:12]. See IPA[47:12] for more details.

**Otherwise:**

Reserved, RES0.
IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1IS instruction

Accesses to this instruction use the following encodings:

TLBI IPAS2E1IS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_IPAS2E1IS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_IPAS2E1IS(X[t]);
The TLBI IPAS2E1OS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI IPAS2E1OS are UNDEFINED.

**Attributes**

TLBI IPAS2E1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI IPAS2E1OS input value bit assignments are:

<p>| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|</p>
<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TTL</th>
<th>RES0</th>
<th>IPA[51:48]</th>
<th>IPA[47:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
</tbody>
</table>

**NS, bit [63]**

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.
When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

**Bits [62:48]**

Reserved, RES0.

**TTL, bits [47:44]**

*When ARMv8.4-TTL is implemented:*

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**Bits [43:40]**

Reserved, RES0.

**IPA[51:48], bits [39:36]**

Extension to IPA[47:12]. See IPA[47:12] for more details.

**IPA[47:12], bits [35:0]**

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

**Executing the TLBI IPAS2E1OS instruction**

Accesses to this instruction use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2E1OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2E1OS(X[t]);

The TLBI IPAS2LE1 characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - $\text{SCR}_{EL3}.\text{NS}==1$ and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - $\text{SCR}_{EL3}.\text{NS}==0$ and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

**Attributes**

TLBI IPAS2LE1 is a 64-bit System instruction.

**Field descriptions**

The TLBI IPAS2LE1 input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NS | RES0 | TTL | RES0 | IPA[51:48] | IPA[47:12] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is $\text{RES}0$, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is $\text{RES}0$. 
Otherwise:

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

**Bits [43:40]**

Reserved, RES0.

**IPA[51:48], bits [39:36]**

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

Otherwise:

Reserved, RES0.
IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1 instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI IPAS2LE1\{, \langle \langle t \rangle \rangle \}}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_IPAS2LE1(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_IPAS2LE1(X[t]);
The TLBI IPAS2LE1IS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

**Attributes**

TLBI IPAS2LE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI IPAS2LE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TTL</th>
<th>RES0</th>
<th>IPA[51:48]</th>
<th>IPA[47:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
</tr>
<tr>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.
When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

**Otherwise:**

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**Bits [43:40]**

Reserved, RES0.

**IPA[51:48], bits [39:36]**

When ARMv8.2-LPA is implemented:

Extension to IPA[47:12]. See IPA[47:12] for more details.

**Otherwise:**

Reserved, RES0.
**IPA[47:12], bits [35:0]**

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

**Executing the TLBI IPAS2LE1IS instruction**

Accesses to this instruction use the following encodings:

```plaintext
TLBI IPAS2LE1IS{, <Xt>}
```

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>
```

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_IPAS2LE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_IPAS2LE1IS(X[t]);
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI IPAS2LE1OS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - \( \text{SCR}_{EL3}.NS==1 \) and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - \( \text{SCR}_{EL3}.NS==0 \) and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI IPAS2LE1OS are UNDEFINED.

**Attributes**

TLBI IPAS2LE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI IPAS2LE1OS input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TTL</th>
<th>RES0</th>
<th>IPA[51:48]</th>
<th>IPA[47:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
</tr>
<tr>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>
When the instruction is executed in Non-secure state, this field is \texttt{RES0}, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is \texttt{RES0}.

**Otherwise:**

Reserved, \texttt{RES0}.

**Bits [62:48]**

Reserved, \texttt{RES0}.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is \texttt{RES0}.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, \texttt{RES0}.

**Bits [43:40]**

Reserved, \texttt{RES0}.

**IPA[51:48], bits [39:36]**

Extension to IPA[47:12]. See IPA[47:12] for more details.

**IPA[47:12], bits [35:0]**

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are \texttt{RES0}.
When ARMv8.2-LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

**Executing the TLBI IPAS2LE1OS instruction**

Accesses to this instruction use the following encodings:

```
TLBI IPAS2LE1OS{, <Xt>}
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_IPAS2LE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_IPAS2LE1OS(X[t]);
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TLBI RIPAS2E1, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI RIPAS2E1 characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - \( \text{SCR}_{EL3}.\text{NS} == 1 \) and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - \( \text{SCR}_{EL3}.\text{NS} == 0 \) and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula \( \text{BaseADDR} < \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^{(5 \times \text{SCALE} +1) \times \text{Translation_Granule Size}}). \)

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If \( \text{TTL}==01 \) and \( \text{BaseADDR}[29:12] \) is not equal to 00000000000000000000000000.
  - If \( \text{TTL}==10 \) and \( \text{BaseADDR}[20:12] \) is not equal to 000000000.
- For the 16K translation granule:
  - If \( \text{TTL}==10 \) and \( \text{BaseADDR}[24:14] \) is not equal to 00000000000.
- For the 64K translation granule:
  - If \( \text{TTL}==01 \) and \( \text{BaseADDR}[41:16] \) is not equal to 000000000000000000000000000000.
  - If \( \text{TTL}==10 \) and \( \text{BaseADDR}[28:16] \) is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2E1 are **UNDEFINED**.

**Attributes**

TLBI RIPAS2E1 is a 64-bit System instruction.
Field descriptions

The TLBI RIPAS2E1 input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
</tr>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 IPA is in the Secure IPA space.</td>
<td></td>
</tr>
<tr>
<td>0b1 IPA is in the Non-secure IPA space.</td>
<td></td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00 Reserved.</td>
<td></td>
</tr>
<tr>
<td>0b01 4K translation granule.</td>
<td></td>
</tr>
<tr>
<td>0b10 16K translation granule.</td>
<td></td>
</tr>
<tr>
<td>0b11 64K translation granule.</td>
<td></td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

### Executing the TLBI RIPAS2E1 instruction

Accesses to this instruction use the following encodings:

TLBI RIPAS2E1{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  UNDEFINED;
elszf PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RIPAS2E1(X[t]);
elszf PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_RIPAS2E1(X[t]);
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI RIPAS2E1IS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - SCR_EL3.NS==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - SCR_EL3.NS==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula \[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)^2(5\times\text{SCALE} +1) \times \text{Translation Granule Size})\].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2E1IS are UNDEFINED.

**Attributes**

TLBI RIPAS2E1IS is a 64-bit System instruction.
Field descriptions

The TLBI RIPAS2E1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

**Otherwise:**

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
### TTL RIPAS2E1IS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

### BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

### Executing the TLBI RIPAS2E1IS instruction

Accesses to this instruction use the following encodings:

`TLBI RIPAS2E1IS{, <Xt>}`

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RIPAS2E1IS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_RIPAS2E1IS(X[t]);
```

27/09/2019 18:22, 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TLBI RIPAS2E1OS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI RIPAS2E1OS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - \texttt{SCR\_EL3.NS==1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - \texttt{SCR\_EL3.NS==0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula \[ \text{BaseADDR} <= \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) * 2^{(5\text{\_SCALE} + 1) \times \text{Translation\_Granule\_Size})].

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is \textit{UNPREDICTABLE} when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2E1OS are \textit{UNDEFINED}.

**Attributes**

TLBI RIPAS2E1OS is a 64-bit System instruction.
Field descriptions

The TLBI RIPAS2E1OS input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td></td>
<td>62</td>
<td></td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td>59</td>
<td></td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td>56</td>
<td></td>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td>53</td>
<td></td>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td>50</td>
<td></td>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td>47</td>
<td></td>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>44</td>
<td></td>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td>41</td>
<td></td>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td>38</td>
<td></td>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td>35</td>
<td></td>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>32</td>
<td></td>
<td>31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>29</td>
<td></td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>26</td>
<td></td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td>23</td>
<td></td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td>20</td>
<td></td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>17</td>
<td></td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>14</td>
<td></td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>11</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>8</td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>5</td>
<td></td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NS, bit [63]

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>
BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1OS instruction

Accesses to this instruction use the following encodings:

TLBI RIPAS2E1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RIPAS2E1OS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_RIPAS2E1OS(X[t]);
The TLBI RIPAS2LE1 characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - \( \text{SCR}_{\text{EL3}}.	ext{NS} = 1 \) and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - \( \text{SCR}_{\text{EL3}}.	ext{NS} = 0 \) and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula \([\text{BaseADDR} <= VA < \text{BaseADDR} + ((\text{NUM} +1)*2^{(5*\text{SCALE} +1) \ast \text{Translation_Granule_Size}})].\)

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1 are UNDEFINED.

**Attributes**

TLBI RIPAS2LE1 is a 64-bit System instruction.
Field descriptions

The TLBI RIPAS2LE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NS, bit [63]

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
### TTL, Meaning

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved. Hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

### BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

### Executing the TLBI RIPAS2LE1 instruction

Accesses to this instruction use the following encodings:

**TLBI RIPAS2LE1{, <Xt>}**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```plaintext
def if PSTATE.EL == EL0 then
    UNDEFINED;
def elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        TLBI_RIPAS2LE1(X[t]);
    elsif PSTATE.EL == EL3 then
        if !EL2Enabled() then
            //no operation
        else
            TLBI_RIPAS2LE1(X[t]);
```

27/09/2019 18:22; 6134483bd14dc8c12a99c84cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI RIPAS2LE1IS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- One of the following applies:
  - \( \text{SCR} \_\text{EL}3\).\text{NS}==1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - \( \text{SCR} \_\text{EL}3\).\text{NS}==0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula \( \text{BaseADDR} <= \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^{(5 \times \text{SCALE} +1) \times \text{Translation Granule Size}}) \).

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1IS are **UNDEFINED**.

**Attributes**

TLBI RIPAS2LE1IS is a 64-bit System instruction.
Field descriptions

The TLBI RIPAS2LE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>RES0</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
<td>57</td>
</tr>
<tr>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
<td>45</td>
<td>44</td>
<td>43</td>
</tr>
<tr>
<td>42</td>
<td>41</td>
<td>40</td>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
<tr>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

** Otherwise:**

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RIPAS2LE1IS instruction**

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2LE1IS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_RIPAS2LE1IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        TLBI_RIPAS2LE1IS(X[t]);
```
TLBI RIPAS2LE1OS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI RIPAS2LE1OS characteristics are:

**Purpose**

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- One of the following applies:
  - $\text{SCR_EL3} . \text{NS} == 1$ and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
  - $\text{SCR_EL3} . \text{NS} == 0$ and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $\text{BaseADDR} <= \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)^2(5^{\text{SCALE}+1})*\text{Translation_Granule_Size})$.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if $\text{SCR_EL3}.\text{EEL2} == 1$, then:

- A PE with $\text{SCR_EL3}.\text{EEL2} == 1$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $\text{SCR_EL3}.\text{EEL2} == 0$.
- A PE with $\text{SCR_EL3}.\text{EEL2} == 0$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $\text{SCR_EL3}.\text{EEL2} == 1$.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If $\text{TTL} == 01$ and $\text{BaseADDR}[29:12]$ is not equal to $00000000000000000000000000$.
  - If $\text{TTL} == 10$ and $\text{BaseADDR}[20:12]$ is not equal to $0000000000$.

- For the 16K translation granule:
  - If $\text{TTL} == 10$ and $\text{BaseADDR}[24:14]$ is not equal to $00000000000000000000000000$.

- For the 64K translation granule:
  - If $\text{TTL} == 01$ and $\text{BaseADDR}[41:16]$ is not equal to $0000000000000000000000000000000000$. 


If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBi is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1OS are UNDEFINED.

**Attributes**

TLBI RIPAS2LE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RIPAS2LE1OS input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NS | RES0 | TG | SCALE | NUM | TTL | BaseADDR |

**NS, bit [63]**

When ARMv8.4-SecEL2 is implemented:

Not Secure. Specifies the IPA space.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IPA is in the Secure IPA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>IPA is in the Non-secure IPA space.</td>
</tr>
</tbody>
</table>

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When ARMv8.4-SecEL2 is not implemented or is disabled in the current Security state, this field is RES0.

**Otherwise:**

Reserved, RES0.

**Bits [62:48]**

Reserved, RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.
**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to</td>
</tr>
<tr>
<td></td>
<td>invalidate are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved.</td>
</tr>
<tr>
<td></td>
<td>Hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

## Executing the TLBI RIPAS2LE1OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RIPAS2LE1OS{, <Xt>}  

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
  UNDEFINED;
else if PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else if PSTATE.EL == EL2 then
  TLBI_RIPAS2LE1OS(X[t]);
else if PSTATE.EL == EL3 then
  if !EL2Enabled() then
    //no operation
  else
    TLBI_RIPAS2LE1OS(X[t]);
```

27/09/2019 18:22; 6134483bd14dc8c12a99e984cb6e3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI RVAAE1 characteristics are:

### Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of `SCR_EL3_NS`:
  - If `HCR_EL2.{E2H, TGE}` is not `{1, 1}`, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If `HCR_EL2.{E2H, TGE}` is `{1, 1}`, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula `[BaseADDR <= VA < BaseADDR + ((NUM +1)*2^(5*SCALE +1) * Translation_Granule_Size)]`. The invalidation applies to the PE that executes this System instruction.

### Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

### Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAAE1 are **UNDEFINED**.

### Attributes

TLBI RVAAE1 is a 64-bit System instruction.
Field descriptions

The TLBI RVAAE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved. RES0.</td>
</tr>
<tr>
<td>47:46</td>
<td>Translation granule size.</td>
</tr>
<tr>
<td>45:44</td>
<td>The exponent element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>43:39</td>
<td>The base element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>38:37</td>
<td>TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.</td>
</tr>
<tr>
<td>36:0</td>
<td>The starting address for the range of the maintenance instruction.</td>
</tr>
</tbody>
</table>

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>Reserved.</td>
</tr>
<tr>
<td>001</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>010</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>011</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].
Executing the TLBI RVAAE1 instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI RVAAE1}_, \langle X_t \rangle
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIRVAEE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAAE1IS(X[t]);
    end if
elsif PSTATE.EL == EL2 then
    TLBI_RVAAE1(X[t]);
else
    TLBI_RVAAE1(X[t]);
end if
The TLBI RVAAE1IS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of \( \text{SCR\_EL3} \).NS:
  - If \( \text{HCR\_EL2} \).{E2H, TGE} is not \{1, 1\}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If \( \text{HCR\_EL2} \).{E2H, TGE} is \{1, 1\}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} <= \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)\times2^{(5\times\text{SCALE} +1)} \times \text{Translation\_Granule\_Size})]\).

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if \( \text{SCR\_EL3.EEL2}==1 \), then:

- A PE with \( \text{SCR\_EL3.EEL2}==1 \) is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with \( \text{SCR\_EL3.EEL2}==0 \).
- A PE with \( \text{SCR\_EL3.EEL2}==0 \) is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with \( \text{SCR\_EL3.EEL2}==1 \).
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is \textit{UNPREDICTABLE} when:

- For the 4K translation granule:
  - If \( \text{TTL}==01 \) and \( \text{BaseADDR}[29:12] \) is not equal to 000000000000000000.
  - If \( \text{TTL}==10 \) and \( \text{BaseADDR}[20:12] \) is not equal to 000000000.
- For the 16K translation granule:
  - If \( \text{TTL}==10 \) and \( \text{BaseADDR}[24:14] \) is not equal to 0000000000.
- For the 64K translation granule:
  - If \( \text{TTL}==01 \) and \( \text{BaseADDR}[41:16] \) is not equal to 00000000000000000000000000.
If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAAE1IS are UNDEFINED.

**Attributes**

TLBI RVAAE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAAE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47-46</td>
<td>Translation granule size</td>
</tr>
<tr>
<td>45-44</td>
<td>The exponent element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>43-39</td>
<td>The base element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>38-37</td>
<td>TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>
BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAAE1IS instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVAAE1IS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HFGITR_EL2.TLBIRVAAE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_RVAAE1IS(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI_RVAAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_RVAAE1IS(X[t]);
TLBI RVAE1OS, TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI RVAE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not \{1, 1\}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is \{1, 1\}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1)\times 2^{(5\times \text{SCALE} + 1)} \times \text{Translation_Granule_Size})]\).

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 0000000000000000000000000000000000.
If \( \text{TTL} = 10 \) and \( \text{BaseADDR}[28:16] \) is not equal to \( 00000000000000 \).

### Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAAE1OS are UNDEFINED.

### Attributes

TLBI RVAAE1OS is a 64-bit System instruction.

### Field descriptions

The TLBI RVAAE1OS input value bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | TG | SCALE | NUM | TTL | BaseADDR |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

#### Bits [63:48]

Reserved, RES0.

#### TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

#### SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

#### NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

#### TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>
BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1OS instruction

Accesses to this instruction use the following encodings:

TLBI RVAAE1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td></td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIRVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI RVAAE1OS(X[t]);
    elsif PSTATE.EL == EL2 then
        TLBI RVAAE1OS(X[t]);
    elsif PSTATE.EL == EL3 then
        TLBI RVAAE1OS(X[t]);
    end

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI RVAALE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.

- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not \{1, 1\}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is \{1, 1\}, the entry would be required to translate the specified VA using the EL2&0 translation regime.

- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

- The entry is within the address range determined by the formula \[[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation_Granule_Size})]\].

The invalidation applies to the PE that executes this System instruction.

**Note**

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL=01 and BaseADDR[29:12] is not equal to 00000000000000000000.
  - If TTL=10 and BaseADDR[20:12] is not equal to 00000000000.

- For the 16K translation granule:
  - If TTL=10 and BaseADDR[24:14] is not equal to 000000000000.

- For the 64K translation granule:
  - If TTL=01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL=10 and BaseADDR[28:16] is not equal to 00000000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAALE1 are UNDEFINED.

**Attributes**

TLBI RVAALE1 is a 64-bit System instruction.
### Field descriptions

The TLBI RVALE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>Reserved</td>
</tr>
<tr>
<td>61</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>Reserved</td>
</tr>
<tr>
<td>55</td>
<td>Reserved</td>
</tr>
<tr>
<td>54</td>
<td>Reserved</td>
</tr>
<tr>
<td>53</td>
<td>Reserved</td>
</tr>
<tr>
<td>52</td>
<td>Reserved</td>
</tr>
<tr>
<td>51</td>
<td>Reserved</td>
</tr>
<tr>
<td>50</td>
<td>Reserved</td>
</tr>
<tr>
<td>49</td>
<td>Reserved</td>
</tr>
<tr>
<td>48</td>
<td>Reserved</td>
</tr>
<tr>
<td>47</td>
<td>Reserved</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

### Bits [63:48]

Reserved, RES0.

### TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

### SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

### NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

### TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate</td>
</tr>
<tr>
<td></td>
<td>are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and</td>
</tr>
<tr>
<td></td>
<td>hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

### BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].
Executing the TLBI RVAALE1 instruction

Accesses to this instruction use the following encodings:

TLBI RVAALE1{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIRVAAL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
    TLBI_RVAAE1IS(X[t]);
  else
    TLBI_RVAALE1(X[t]);
elsif PSTATE.EL == EL2 then
  TLBI_RVAALE1(X[t]);
else
  TLBI_RVAALE1(X[t]);
The TLBI RVAALE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)\times2^{(5\times\text{SCALE} +1) \times \text{Translation_Granule_Size}})\].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 0000000000000000.
If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAALE1IS are UNDEFINED.

**Attributes**

TLBI RVAALE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAALE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47:46</td>
<td>Translation granule size.</td>
</tr>
<tr>
<td>45:44</td>
<td>The exponent element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>43:39</td>
<td>The base element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>38:37</td>
<td>TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granularity size for the translations that are being invalidated. If the translations used a different translation granularity size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granularity, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granularity, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>
BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1IS instruction

Accesses to this instruction use the following encodings:

TLBI RVAALE1IS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIRVAALE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_RVAALE1IS(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI_RVAALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_RVAALE1IS(X[t]);
TLBI RVAALE1OS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI RVAALE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^{(5 \times \text{SCALE} +1) \times \text{Translation_Granule_Size}}))\).

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both:

- Global entries.
- Non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAALE1OS are UNDEFINED.

**Attributes**

TLBI RVAALE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAALE1OS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-48</td>
<td>Reserved</td>
</tr>
<tr>
<td>47-46</td>
<td>Translation granule size.</td>
</tr>
<tr>
<td>45-44</td>
<td>The exponent element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>43-39</td>
<td>The base element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>38-37</td>
<td>TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>
BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1OS instruction

Accesses to this instruction use the following encodings:

$$\text{TLBI RVAALE1OS}, \langle X_t \rangle$$

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIRVAALE1OS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_RVAALE1OS(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI_RVAALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_RVAALE1OS(X[t]);
TLBI RVAE1, TLB Range Invalidate by VA, EL1

The TLBI RVAE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3:NS:
  - If \( \text{HCR\_EL2.\{E2H, TGE\}} \) is not \( \{1, 1\} \), the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If \( \text{HCR\_EL2.\{E2H, TGE\}} \) is \( \{1, 1\} \), the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} <= \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)\times 2^{(5\times \text{SCALE} +1)} \times \text{Translation\_Granule\_Size})]\).

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE1 are **UNDEFINED**.

**Attributes**

TLBI RVAE1 is a 64-bit System instruction.
Field descriptions

The TLBI RVAE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ASID</td>
</tr>
<tr>
<td>62</td>
<td>TG</td>
</tr>
<tr>
<td>61</td>
<td>SCALE</td>
</tr>
<tr>
<td>60</td>
<td>NUM</td>
</tr>
<tr>
<td>59</td>
<td>TTL</td>
</tr>
<tr>
<td>58</td>
<td>BaseADDR</td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>BaseADDR</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAE1 instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI RVAE1}{, \langle X_t \rangle}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIRVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        TLBI_RVAE1IS(X[t]);
    else
        TLBI_RVAE1(X[t]);
    endif
elsif PSTATE.EL == EL2 then
    TLBI_RVAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVAE1(X[t]);
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI RVAE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If $HCR_{EL2}.\{E2H, TGE\}$ is not $\{1, 1\}$, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If $HCR_{EL2}.\{E2H, TGE\}$ is $\{1, 1\}$, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)^2 \cdot (5\cdot\text{SCALE} + 1) \cdot \text{Translation Granule Size})]$.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with $\text{SCR}_E\text{L3}.\text{EEL2}=\text{1}$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $\text{SCR}_E\text{L3}.\text{EEL2}=\text{0}$.
- A PE with $\text{SCR}_E\text{L3}.\text{EEL2}=\text{0}$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $\text{SCR}_E\text{L3}.\text{EEL2}=\text{1}$.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.

If TTL==10 and BaseADDR[28:16] is not equal to 000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE1IS are UNDEFINED.

**Attributes**

TLBI RVAE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>[63:48]</td>
<td>ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction. Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field. If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.</td>
</tr>
<tr>
<td>TG</td>
<td>[47:46]</td>
<td>Translation granule size. The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.</td>
</tr>
<tr>
<td>SCALE</td>
<td>[45:44]</td>
<td>The exponent element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>NUM</td>
<td>[43:39]</td>
<td>The base element of the calculation that is used to produce the upper range.</td>
</tr>
<tr>
<td>TTL</td>
<td>[38:37]</td>
<td>TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseADDR</td>
<td>[32:0]</td>
<td>Base Address</td>
</tr>
<tr>
<td>TTL</td>
<td>Meaning</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
<td></td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
<td></td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
<td></td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAE1IS instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVAE1IS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVAE1IS(X[t]);
    elsif PSTATE_EL == EL2 then
        TLBI_RVAE1IS(X[t]);
    elsif PSTATE_EL == EL3 then
        TLBI_RVAE1IS(X[t]);
```
The TLBI RVAE1OS characteristics are:

**Purpose**

Invalidate cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation Granule Size})\].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.

If TTL==10 and BaseADDR[28:16] is not equal to 00000000000000.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE1OS are UNDEFINED.

Attributes

TLBI RVAE1OS is a 64-bit System instruction.

Field descriptions

The TLBI RVAE1OS input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
**TLI RVAE1OS, TLB Range Invalidate by VA, EL1, Outer Shareable**

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to</td>
</tr>
<tr>
<td></td>
<td>invalidate are Level 1 translation table entries. When using a</td>
</tr>
<tr>
<td></td>
<td>16KB translation granule, this value is reserved and hardware should</td>
</tr>
<tr>
<td></td>
<td>treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAE1OS instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVAE10S{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TTBOS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBI_RVAE1OS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_RVAE1OS(X[t]);
  elsif PSTATE.EL == EL2 then
    TLBI_RVAE1OS(X[t]);
  elsif PSTATE.EL == EL3 then
    TLBI_RVAE1OS(X[t]);
```
TLBI RVAE2, TLB Range Invalidate by VA, EL2

The TLBI RVAE2 characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula
  \[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation Granule Size})\], using the EL2 or EL2&0 translation regime.
- If \text{HCR\_EL2\_E2H} == 0, the entry is from any level of the translation table walk.
- If \text{HCR\_EL2\_E2H} == 1, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE2 are UNDEFINED.

Attributes

TLBI RVAE2 is a 64-bit System instruction.

Field descriptions

The TLBI RVAE2 input value bit assignments are:
### TLBI RVAE2, TLB Range Invalidate by VA, EL2

<table>
<thead>
<tr>
<th>ASID</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

### ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

### TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

### SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

### NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

### TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

### BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAE2 instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVAE2{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RVAE2(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_RVAE2(X[t]);
TLBI RVAE2IS, TLB Range Invalidate by VA, EL2, Inner Shareable

The TLBI RVAE2IS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula 
  \[ \text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)\times2^{(5\times\text{SCALE} +1) \times \text{Translation_Granule_Size}}) \], using the EL2 or EL2&0 translation regime.
- If \[ \text{HCR}_{\text{EL2}} \cdot E2H == 0 \], the entry is from any level of the translation table walk.
- If \[ \text{HCR}_{\text{EL2}} \cdot E2H == 1 \], one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 00000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:12] is not equal to 0000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE2IS are UNDEFINED.

**Attributes**

TLBI RVAE2IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAE2IS input value bit assignments are:
ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAE2IS instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVAE2IS{
  , <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RVAE2IS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_RVAE2IS(X[t]);
```
TLBI RVAE2OS, TLB Range Invalidate by VA, EL2, Outer Shareable

The TLBI RVAE2OS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation Granule Size})]$, using the EL2 or EL2&0 translation regime.
- If $\text{HCR}_{\text{EL2}}.E2H == 0$, the entry is from any level of the translation table walk.
- If $\text{HCR}_{\text{EL2}}.E2H == 1$, one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and \text{BaseADDR}[29:12] is not equal to 000000000000000.
  - If TTL==10 and \text{BaseADDR}[20:12] is not equal to 00000000.
- For the 16K translation granule:
  - If TTL==10 and \text{BaseADDR}[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and \text{BaseADDR}[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and \text{BaseADDR}[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE2OS are **UNDEFINED**.

**Attributes**

TLBI RVAE2OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAE2OS input value bit assignments are:
**ASID, bits [63:48]**

When HCR_EL2.E2H == 1:

- ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.
- Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
- If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are `RES0`.

Otherwise:

- Reserved, `RES0`.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

### Executing the TLBI RVAE2OS instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI RVAE2OS\{, } <Xt>\}\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsf if PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsf if PSTATE.EL == EL2 then
  TLBI_VRAE2OS(X[t]);
elsf if PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_VRAE2OS(X[t]);
TLBI RVAE3, TLB Range Invalidate by VA, EL3

The TLBI RVAE3 characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula \[ \text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)\times2^{(5\times\text{SCALE} +1)} \times \text{Translation_Granule_Size}) \].

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is *UNPREDICTABLE* when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE3 are *UNDEFINED*.

**Attributes**

TLBI RVAE3 is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAE3 input value bit assignments are:

|   | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | TG | SCALE | NUM | TTL | BaseADDR |

**Bits [63:48]**

Reserved, RES0.
**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVAE3 instruction**

Accesses to this instruction use the following encodings:

TLBI RVAE3{, <Xt>}

<table>
<thead>
<tr>
<th></th>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI RVAE3(X[t]);
The TLBI RVAE3IS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula \[ \text{BaseADDR} \leq VA < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation Granule Size}) \].

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 00000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 00000000000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE3IS are **UNDEFINED**.

**Attributes**

TLBI RVAE3IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAE3IS input value bit assignments are:

|   | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| TG  |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| SCALE |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| NUM |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| TTL |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| BaseADDR |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

**Bits [63:48]**

Reserved, RES0.
TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3IS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE3IS{<Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b11</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_RVAE3IS(X[t]);
The TLBI RVAE3OS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula 
  \[ \text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^{(5 \times \text{SCALE} +1) \times \text{Translation Granule Size}}) \].

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVAE3OS are **UNDEFINED**.

**Attributes**

TLBI RVAE3OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVAE3OS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-48</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.
TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to</td>
</tr>
<tr>
<td></td>
<td>invalidate are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and</td>
</tr>
<tr>
<td></td>
<td>hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3OS instruction

Accesses to this instruction use the following encodings:

```
TLBI RVAE3OS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_RVAE3OS(X[t]);
TLBI RVALE1, TLB Range Invalidate by VA, Last level, EL1

The TLBI RVALE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of $SCR_EL3.NS$:
  - If $HCR_EL2\{E2H, TGE\}$ is not $\{1,1\}$, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If $HCR_EL2\{E2H, TGE\}$ is $\{1,1\}$, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + \left( (\text{NUM} + 1) \times 2^\left(5 \times \text{SCALE} + 1\right) \times \text{Translation Granule Size}\right)]$.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 0000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 00000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000.

For more information about the architectural requirements for this System instruction see 'Invalidation of TLB entries from stage 2 translations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE1 are **UNDEFINED**.

**Attributes**

TLBI RVALE1 is a 64-bit System instruction.
Field descriptions

The TLBI RVALE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>ASID</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
</tr>
<tr>
<td>51</td>
<td>50</td>
<td>49</td>
<td>48</td>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>43</td>
<td>42</td>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVALE1 instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVALE1{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIRVALE1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_RVALE1IS(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI_RVALE1(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_RVALE1(X[t]);
TLBI RVALE1IS, TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI RVALE1IS characteristics are:

**Purpose**

Invalidate cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \( [\text{BaseADDR} \leq VA < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation\_Granule\_Size})] \).

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

### Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE1IS are UNDEFINED.

### Attributes

TLBI RVALE1IS is a 64-bit System instruction.

### Field descriptions

The TLBI RVALE1IS input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ASID | TG | SCALE | NUM | TTL | BaseADDR |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
**TLBI RVALE1IS, TLB Range Invalidate by VA, Last level, EL1, Inner Shareable**

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVALE1IS instruction**

Accesses to this instruction use the following encodings:

`TLBI RVALE1IS{}, <Xt>`

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTL == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIRVALE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_RVALE1IS(X[t]);
  endif;
elsif PSTATE.EL == EL2 then
  TLBI_RVALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_RVALE1IS(X[t]);
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TLBI RVALE1OS, TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI RVALE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{5 \times \text{SCALE} + 1} \times \text{Translation Granule Size})]\).

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE1OS are **UNDEFINED**.

**Attributes**

TLBI RVALE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE1OS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Positions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-48</td>
<td>ASID</td>
</tr>
<tr>
<td>47-46</td>
<td>TG</td>
</tr>
<tr>
<td>45-44</td>
<td>SCALE</td>
</tr>
<tr>
<td>43-39</td>
<td>NUM</td>
</tr>
<tr>
<td>38-37</td>
<td>TTL</td>
</tr>
<tr>
<td>31-0</td>
<td>BaseADDR</td>
</tr>
</tbody>
</table>

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are **RES0**.

**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.
### TTL, Meaning

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

#### BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

### Executing the TLBI RVALE1OS instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI RVALE1OS\{, <Xt>\}}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_RVALE1OS(X[t]);
    end
elsif PSTATE.EL == EL2 then
    TLBI_RVALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_RVALE1OS(X[t]);
The TLBI RVALE2 characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula
  \[
  \text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{5 \times \text{SCALE} + 1} \times \text{Translation_Granule_Size})
  \]
  using the EL2 or EL2&0 translation regime.
- If \( \text{HCR}_{EL2}.E2H = 0 \), the entry is from the final level of the translation table walk.
- If \( \text{HCR}_{EL2}.E2H = 1 \), one of the following applies:
  - The entry is a global entry from the final level of translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 0000000000000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 00000000000000000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE2 are **UNDEFINED**.

**Attributes**

TLBI RVALE2 is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE2 input value bit assignments are:
ASID, bits [63:48]

When HCR_EL2.E2H == 1:

- ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.
- Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
- If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

- Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to</td>
</tr>
<tr>
<td></td>
<td>invalidate are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and</td>
</tr>
<tr>
<td></td>
<td>hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVALE2 instruction**

Accesses to this instruction use the following encodings:

TLBI RVALE2{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RVALE2(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_RVALE2(X[t]);
The TLBI RVALE2IS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[BaseADDR <= VA < BaseADDR + ((NUM +1)*2^{(5*SCALE +1) * Translation_Granule_Size})$ using the EL2 or EL2&0 translation regime.
- If $HCR_{EL2}.E2H == 0$, the entry is from the final level of the translation table walk.
- If $HCR_{EL2}.E2H == 1$, one of the following applies:
  - The entry is a global entry from the final level of translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 00000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE2IS are **UNDEFINED**.

**Attributes**

TLBI RVALE2IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE2IS input value bit assignments are:
ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate</td>
</tr>
<tr>
<td></td>
<td>are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and</td>
</tr>
<tr>
<td></td>
<td>hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is `BaseADDR[48:12]`.
When using a 16KB translation granule, this field is `BaseADDR[50:14]`.
When using a 64KB translation granule, this field is `BaseADDR[52:16]`.

**Executing the TLBI RVALE2IS instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI RVALE2IS}\{X_t\}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if \( \text{PSTATE.EL} == \text{EL0} \) then
UNDEFINED;
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
  if \( \text{EL2Enabled}() \) && \( \text{HCR_EL2.NV} == '1' \) then
    \text{AArch64.SystemAccessTrap(EL2, 0x18)};
  else
    UNDEFINED;
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
  \text{TLBI_RVALE2IS(X[t])};
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
  if !\text{EL2Enabled}() then
    UNDEFINED;
  else
    \text{TLBI_RVALE2IS(X[t])};
TLBI RVALE2OS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI RVALE2OS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula
  \[ \text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation_Granule_Size}) \] using the EL2 or EL2&0 translation regime.
- If \( \text{HCR_EL2.E2H} = 0 \), the entry is from the final level of the translation table walk.
- If \( \text{HCR_EL2.E2H} = 1 \), one of the following applies:
  - The entry is a global entry from the final level of translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE2OS are **UNDEFINED**.

**Attributes**

TLBI RVALE2OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE2OS input value bit assignments are:
ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reserved</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVALE2OS instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI RVALE2OS\{} \substack{<Xt>}\}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_RVALE2OS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_RVALE2OS(X[t]);
TLBI RVALE3, TLB Range Invalidate by VA, Last level, EL3

The TLBI RVALE3 characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} +1) \times 2^5 \times \text{SCALE} + 1) \times \text{Translation Granule Size})\].

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If \(\text{TTL}==01\) and \(\text{BaseADDR}[29:12]\) is not equal to \(00000000000000000000\).
  - If \(\text{TTL}==10\) and \(\text{BaseADDR}[20:12]\) is not equal to \(0000000000\).

- For the 16K translation granule:
  - If \(\text{TTL}==10\) and \(\text{BaseADDR}[24:14]\) is not equal to \(0000000000\).

- For the 64K translation granule:
  - If \(\text{TTL}==01\) and \(\text{BaseADDR}[41:16]\) is not equal to \(00000000000000000000000000\).
  - If \(\text{TTL}==10\) and \(\text{BaseADDR}[28:16]\) is not equal to \(000000000000\).

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE3 are **UNDEFINED**.

**Attributes**

TLBI RVALE3 is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE3 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>TG</td>
</tr>
<tr>
<td>61</td>
<td>SCALE</td>
</tr>
<tr>
<td>60</td>
<td>NUM</td>
</tr>
<tr>
<td>59</td>
<td>TTL</td>
</tr>
<tr>
<td>58</td>
<td>BaseADDR</td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.
TG, bits [47:46]

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate</td>
</tr>
<tr>
<td></td>
<td>are Level 1 translation table entries.</td>
</tr>
<tr>
<td></td>
<td>When using a 16KB translation granule, this value is reserved and</td>
</tr>
<tr>
<td></td>
<td>hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3 instruction

Accesses to this instruction use the following encodings:

TLBI RVALE3{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b11</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b10</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVALE3(X[t]);
TLBI RVALE3IS, TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI RVALE3IS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) \times 2^{(5 \times \text{SCALE} + 1)} \times \text{Translation Granule Size})]$.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.

- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 0000000000.

- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 0000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE3IS are **UNDEFINED**.

**Attributes**

TLBI RVALE3IS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE3IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-48</td>
<td>RES0</td>
</tr>
<tr>
<td>47-45</td>
<td>TG</td>
</tr>
<tr>
<td>44-32</td>
<td>SCALE</td>
</tr>
<tr>
<td>31-16</td>
<td>NUM</td>
</tr>
<tr>
<td>15-0</td>
<td>TTL</td>
</tr>
<tr>
<td>31</td>
<td>BaseADDR</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.
TG, bits [47:46]
Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]
The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]
The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]
TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

BaseADDR, bits [36:0]
The starting address for the range of the maintenance instruction.
When using a 4KB translation granule, this field is BaseADDR[48:12].
When using a 16KB translation granule, this field is BaseADDR[50:14].
When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3IS instruction
Accesses to this instruction use the following encodings:

TLBI RVALE3IS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVALE3IS(X[t]);
The TLBI RVALE3OS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula \([\text{BaseADDR} <= \text{VA} < \text{BaseADDR} + ((\text{NUM} +1)*2^((5\times\text{SCALE} +1) * \text{Translation Granule Size}))]\).

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is **UNPREDICTABLE** when:

- For the 4K translation granule:
  - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000000000.
  - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
  - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
  - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000.
  - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI RVALE3OS are **UNDEFINED**.

**Attributes**

TLBI RVALE3OS is a 64-bit System instruction.

**Field descriptions**

The TLBI RVALE3OS input value bit assignments are:

<table>
<thead>
<tr>
<th>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</th>
<th>RES0</th>
<th>TG</th>
<th>SCALE</th>
<th>NUM</th>
<th>TTL</th>
<th>BaseADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>BaseADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.
**TG, bits [47:46]**

Translation granule size.

<table>
<thead>
<tr>
<th>TG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b01</td>
<td>4K translation granule.</td>
</tr>
<tr>
<td>0b10</td>
<td>16K translation granule.</td>
</tr>
<tr>
<td>0b11</td>
<td>64K translation granule.</td>
</tr>
</tbody>
</table>

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

**SCALE, bits [45:44]**

The exponent element of the calculation that is used to produce the upper range.

**NUM, bits [43:39]**

The base element of the calculation that is used to produce the upper range.

**TTL, bits [38:37]**

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The entries in the range can be using any level for the translation</td>
</tr>
<tr>
<td></td>
<td>table entries.</td>
</tr>
<tr>
<td>0b01</td>
<td>When using a 4KB or 64KB translation granule, all entries to invalidate are Level 1 translation table entries. When using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00.</td>
</tr>
<tr>
<td>0b10</td>
<td>All entries to invalidate are Level 2 translation table entries.</td>
</tr>
<tr>
<td>0b11</td>
<td>All entries to invalidate are Level 3 translation table entries.</td>
</tr>
</tbody>
</table>

**BaseADDR, bits [36:0]**

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

**Executing the TLBI RVALE3OS instruction**

Accesses to this instruction use the following encodings:

```
TLBI RVALE3OS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b101</td>
</tr>
</tbody>
</table>

Page 1645
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_RVALE3OS(X[t]);
The TLBI VAAE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Note**

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VAAE1 is a 64-bit System instruction.

**Field descriptions**

The TLBI VAAE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>RES0, TTL, VA[55:12]</td>
</tr>
<tr>
<td>31-0</td>
<td>VA[55:12]</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAAE1 instruction**

Accesses to this instruction use the following encodings:

```
TLBI VAAE1{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b011</td>
</tr>
</tbody>
</table>

Page 1648
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VAAE1IS(X[t]);
    else
        TLBI_VAAE1(X[t]);
    end if;
elsif PSTATE.EL == EL2 then
    TLBI_VAAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAAE1(X[t]);
else
    TLBI_VAAE1IS(X[t]);
end if;

TLBI VAAE1, TLB Invalidate by VA, All ASID, EL1
TLBI VAAE1IS, TLB Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI VAAE1IS characteristics are:

**Purpose**

Invalidate cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VAAE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAAE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>TTL</td>
</tr>
<tr>
<td>55</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>31</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>30</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>29</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>28</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>27</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>26</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>25</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>24</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>23</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>22</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>21</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>20</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>19</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>18</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>17</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>16</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>15</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>14</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>13</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>12</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>11</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>10</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>9</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>8</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>7</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>6</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>5</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>4</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>3</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>2</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>1</td>
<td>VA[31:12]</td>
</tr>
<tr>
<td>0</td>
<td>VA[31:12]</td>
</tr>
</tbody>
</table>
Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1IS instruction

Accesses to this instruction use the following encodings:
TLBI_VAAE1IS{

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b01 & 0b000 & 0b1000 & 0b0011 & 0b011 \\
\hline
\end{array}
\]

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVAAE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VAAE1IS(X[t]);
  end
elsif PSTATE.EL == EL2 then
  TLBI_VAAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_VAAE1IS(X[t]);

27/09/2019 18:22; 6134483bd14dc8c12a99c984cb6e3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI VAAE1OS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of $SCR_{EL3}.NS$:
  - If $HCR_{EL2}.\{E2H, TGE\}$ is not \{1, 1\}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If $HCR_{EL2}.\{E2H, TGE\}$ is \{1, 1\}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if $SCR_{EL3}.EEL2==1$, then:

- A PE with $SCR_{EL3}.EEL2==1$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $SCR_{EL3}.EEL2==0$.
- A PE with $SCR_{EL3}.EEL2==0$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $SCR_{EL3}.EEL2==1$.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

Configuration

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAAE1OS are UNDEFINED.

Attributes

TLBI VAAE1OS is a 64-bit System instruction.

Field descriptions

The TLBI VAAE1OS input value bit assignments are:
Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.
Executing the TLBI VAAE1OS instruction

Accesses to this instruction use the following encodings:

TLBI VAAE1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVAAE1OS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VAAE1OS(X[t]);
  elsif PSTATE.EL == EL2 then
    TLBI_VAAE1OS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_VAAE1OS(X[t]);
The TLBI VAALE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3:NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Note**

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VAALE1 is a 64-bit System instruction.

**Field descriptions**

The TLBI VAALE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:48]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TTL, bits [47:44]</strong></td>
<td></td>
</tr>
<tr>
<td>When ARMv8.4-TTL is implemented:</td>
<td></td>
</tr>
<tr>
<td>Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.</td>
<td></td>
</tr>
</tbody>
</table>
TTL | Meaning
---|---
0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is res0.
0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL<3:2> is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.
0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL<3:2> is 0b00. 0b01: Reserved. Treat as if TTL<3:2> is 0b00. 0b10: Level 2. 0b11: Level 3.
0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL<3:2> is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, res0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as res0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are res0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are res0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAALE1 instruction**

Accesses to this instruction use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b111</td>
</tr>
</tbody>
</table>

TLBI VAALE1{, <Xt>}
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VAALE1IS(X[t]);
    else
        TLBI_VAALE1(X[t]);
    end
elsif PSTATE.EL == EL2 then
    TLBI_VAALE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAALE1(X[t]);
TLBI VAALE1IS, TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI VAALE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not \{1, 1\}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is \{1, 1\}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VAALE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAALE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>63</td>
</tr>
<tr>
<td>TTL</td>
<td>62</td>
</tr>
<tr>
<td>VA[55:12]</td>
<td>61 to 55</td>
</tr>
</tbody>
</table>
**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

**When ARMv8.4-TTL is implemented:**

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAALE1IS instruction**

Accesses to this instruction use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVAALE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VAALE1IS(X[t]);
  end
elsif PSTATE.EL == EL2 then
  TLBI_VAALE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_VAALE1IS(X[t]);
TLBI VAALE1OS, TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI VAALE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of `SCR_EL3.NS`:
  - If `HCR_EL2.{E2H, TGE}` is not `1, 1`, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If `HCR_EL2.{E2H, TGE}` is `1, 1`, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if `SCR_EL3.EEL2==1`, then:

- A PE with `SCR_EL3.EEL2==1` is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with `SCR_EL3.EEL2==0`.
- A PE with `SCR_EL3.EEL2==0` is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with `SCR_EL3.EEL2==1`.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of `0`.

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAALE1OS are **UNDEFINED**.

**Attributes**

TLBI VAALE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAALE1OS input value bit assignments are:
**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

*When ARMv8.4-TTL is implemented:*

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.
Executing the TLBI VAALE1OS instruction

Accesses to this instruction use the following encodings:

TLBI VAALE1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAALE1OS(X[t]);
    endif
elsif PSTATE.EL == EL2 then
    TLBI_VAALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAALE1OS(X[t]);
TLBI VAE1, TLB Invalidate by VA, EL1

The TLBI VAE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of `SCR_EL3 NS`:
  - If `HCR_EL2 {E2H, TGE}` is not `{1, 1}`, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If `HCR_EL2 {E2H, TGE}` is `{1, 1}`, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VAE1 is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE1 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ASID</td>
</tr>
<tr>
<td>62</td>
<td>TTL</td>
</tr>
<tr>
<td>59</td>
<td>VA[55:12]</td>
</tr>
</tbody>
</table>

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are `RES0`.  

---

Page 1665
TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits [55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits [55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1 instruction

Accesses to this instruction use the following encodings:

TLBI VAE1{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.FB == '1' then
        TLBI_VAE1IS(X[t]);
    else
        TLBI_VAE1(X[t]);
    end
elsif PSTATE.EL == EL2 then
    TLBI_VAE1(X[t]);
elsif PSTATE.EL == EL3 then
    TLBI_VAE1(X[t]);
The TLBI VAE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of \( \text{SCR_EL3.NS} \):
  - If \( \text{HCR_EL2.\{E2H, TGE\}} \) is not \( \{1, 1\} \), the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If \( \text{HCR_EL2.\{E2H, TGE\}} \) is \( \{1, 1\} \), the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if \( \text{SCR_EL3.EEL2} == 1 \), then:

- A PE with \( \text{SCR_EL3.EEL2} == 1 \) is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with \( \text{SCR_EL3.EEL2} == 0 \).
- A PE with \( \text{SCR_EL3.EEL2} == 0 \) is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with \( \text{SCR_EL3.EEL2} == 1 \).
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

**Configuration**

**Attributes**

TLBI VAE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE1IS input value bit assignments are:
ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are **RES0**.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is <strong>RES0</strong>.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, **RES0**.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as **RES0**.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
• Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.

• Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE1IS instruction**

Accesses to this instruction use the following encodings:

```plaintext
TLBI VAE1IS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVAE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VAE1IS(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI_VAE1IS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_VAE1IS(X[t]);
```
TLBI VAE1OS, TLB Invalidate by VA, EL1, Outer Shareable

The TLBI VAE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAE1OS are UNDEFINED.

**Attributes**

TLBI VAE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE1OS input value bit assignments are:
ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are \texttt{RES0}.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is \texttt{RES0}.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, \texttt{RES0}.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as \texttt{RES0}.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
• Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.

• Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE10S instruction**

Accesses to this instruction use the following encodings:

TLBI VAE10S{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGITR_EL2.TLBIVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI_VAE10S(X[t]);
    elsif PSTATE.EL == EL2 then
        TLBI_VAE10S(X[t]);
    elsif PSTATE.EL == EL3 then
        TLBI_VAE10S(X[t]);
```
The TLBI VAE2 characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be required to translate the specified VA using the EL2 or the EL2&0 translation regime.
- If \texttt{HCR\_E2H} == 0, the entry is from any level of the translation table walk.
- If \texttt{HCR\_E2H} == 1 one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VAE2 is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE2 input value bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are \texttt{RES0}.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE2 instruction**

Accesses to this instruction use the following encodings:

**TLBI VAE2{, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VAE2(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VAE2(X[t]);
The TLBI VAE2IS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be required to translate the specified VA using the EL2 or the EL2&0 translation regime.
- If $\text{HCR}_\text{EL2}.E2H == 0$, the entry is from any level of the translation table walk.
- If $\text{HCR}_\text{EL2}.E2H == 1$ one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VAE2IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE2IS input value bit assignments are:

<p>| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|</p>
<table>
<thead>
<tr>
<th>ASID</th>
<th>TTL</th>
<th>VA[55:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
</tbody>
</table>

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.
**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01 : Level 1.</td>
</tr>
<tr>
<td></td>
<td>0b10 : Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b10 : Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01 : Level 1.</td>
</tr>
<tr>
<td></td>
<td>0b10 : Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE2IS instruction**

Accesses to this instruction use the following encodings:

```
TLBI VAE2IS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_VAE2IS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_VAE2IS(X[t]);
The TLBI VAE2OS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be required to translate the specified VA using the EL2 or the EL2&0 translation regime.
- If \( HCR\_EL2.E2H = 0 \), the entry is from any level of the translation table walk.
- If \( HCR\_EL2.E2H = 1 \) one of the following applies:
  - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of the translation table walk and matches the specified ASID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAE2OS are UNDEFINED.

**Attributes**

TLBI VAE2OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE2OS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-62</td>
<td>ASID</td>
</tr>
<tr>
<td>61-59</td>
<td>TTL</td>
</tr>
<tr>
<td>58-50</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**ASID, bits [63:48]**

When \( HCR\_EL2.E2H = 1 \):

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.
Otherwise:

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01 : Level 1.</td>
</tr>
<tr>
<td></td>
<td>0b10 : Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b10 : Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01 : Level 1.</td>
</tr>
<tr>
<td></td>
<td>0b10 : Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE2OS instruction**

Accesses to this instruction use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VAE2OS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VAE2OS(X[t]);
    endif;
endif;

TLBI VAE2OS{", <X[t>]}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI VAE3 characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VAE3 is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE3 input value bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| TTL | VA[55:12] |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
TTL | Meaning
---|---
0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is res0.
0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
| 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.
| 0b01 : Level 1.
| 0b10 : Level 2.
| 0b11 : Level 3.
0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
| 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.
| 0b01 : Reserved. Treat as if TTL<3:2> is 0b00.
| 0b10 : Level 2.
| 0b11 : Level 3.
0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as:
| 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.
| 0b01 : Level 1.
| 0b10 : Level 2.
| 0b11 : Level 3.

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, res0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as res0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are res0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are res0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE3 instruction**

Accesses to this instruction use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VAE3(X[t]);
The TLBI VAE3IS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VAE3IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE3IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Positions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>RES0</td>
</tr>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>VA[12:1]</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
### TTL Meaning

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01: Level 1.</td>
</tr>
<tr>
<td></td>
<td>0b10: Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b10: Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as:</td>
</tr>
<tr>
<td></td>
<td>0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00.</td>
</tr>
<tr>
<td></td>
<td>0b01: Level 1.</td>
</tr>
<tr>
<td></td>
<td>0b10: Level 2.</td>
</tr>
<tr>
<td></td>
<td>0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

### Executing the TLBI VAE3IS instruction

Accesses to this instruction use the following encodings:

**TLBI VAE3IS({, <Xt>])**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b11</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_VAE3IS(X[t]);
The TLBI VAE3OS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VAE3OS are **UNDEFINED**.

**Attributes**

TLBI VAE3OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VAE3OS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49-44</td>
<td>Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.</td>
</tr>
<tr>
<td>43-36</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>35-21</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>20-10</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>9-0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VAE3OS instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI VAE3OS}\{, \langle Xt\rangle\}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VAE3OS(X[t]);
TLBI VALE1, TLB Invalidate by VA, Last level, EL1

The TLBI VALE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VALE1 is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE1 input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ASID | TTL | VA[55:12] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
TTL

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
</tr>
<tr>
<td>0b01xx</td>
</tr>
<tr>
<td>0b10xx</td>
</tr>
<tr>
<td>0b11xx</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VALE1 instruction**

Accesses to this instruction use the following encodings:

\[
\text{TLBI VALE1}, \langle \text{Xt} \rangle
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVALE1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x10);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
    TLBI_VALE1IS(X[t]);
  else
    TLBI_VAAE1(X[t]);
  endif
elsif PSTATE.EL == EL2 then
  TLBI_VAAE1(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_VAAE1(X[t]);
TLBI VALE1IS, TLB Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI VALE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

---

**Note**

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

---

**Configuration**

**Attributes**

TLBI VALE1IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE1IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>ASID</th>
<th>TTL</th>
<th>VA[55:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
</tbody>
</table>
ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.
Executing the TLBI VALE1IS instruction

Accesses to this instruction use the following encodings:

TLBI VALE1IS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGETen == '1') &&
        HFGITR_EL2.TLBIVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TLBI VALE1IS(X[t]);
    elseif PSTATE.EL == EL2 then
        TLBI VALE1IS(X[t]);
    elseif PSTATE.EL == EL3 then
        TLBI VALE1IS(X[t]);

TLBI VALE1OS, TLB Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI VALE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the Security state described by the current value of \( \text{SCR_EL3.NS} \):
  - If \( \text{HCR_EL2.E2H, TGE} \) is not \( \{1, 1\} \), the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If \( \text{HCR_EL2.E2H, TGE} \) is \( \{1, 1\} \), the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if \( \text{SCR_EL3.EEL2}==1 \), then:

- A PE with \( \text{SCR_EL3.EEL2}==1 \) is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with \( \text{SCR_EL3.EEL2}==0 \).
- A PE with \( \text{SCR_EL3.EEL2}==0 \) is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with \( \text{SCR_EL3.EEL2}==1 \).
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VALE1OS are **UNDEFINED**.

**Attributes**

TLBI VALE1OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE1OS input value bit assignments are:
**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
</tbody>
</table>
| 0b01xx| The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as:  
|       | 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.  
|       | 0b01 : Level 1.  
|       | 0b10 : Level 2.  
|       | 0b11 : Level 3. |
| 0b10xx| The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as:  
|       | 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.  
|       | 0b01 : Reserved. Treat as if TTL<3:2> is 0b00.  
|       | 0b10 : Level 2.  
|       | 0b11 : Level 3. |
| 0b11xx| The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as:  
|       | 0b00 : Reserved. Treat as if TTL<3:2> is 0b00.  
|       | 0b01 : Level 1.  
|       | 0b10 : Level 2.  
|       | 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.

Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VALE1OS instruction**

Accesses to this instruction use the following encodings:

TLBI VALE1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBOS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVALE1OS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VALE1OS(X[t]);
  endif;
elsif PSTATE.EL == EL2 then
  TLBI_VALE1OS(X[t]);
elsif PSTATE.EL == EL3 then
  TLBI_VALE1OS(X[t]);
The TLBI VALE2 characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime.
- If $\text{HCR}_\text{EL2}.\text{E2H} = 0$, the entry is from the final level of the translation table walk.
- If $\text{HCR}_\text{EL2}.\text{E2H} = 1$, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of translation table walk that matches the specified ASID.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VALE2 is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE2 input value bit assignments are:

<p>| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|</p>
<table>
<thead>
<tr>
<th>ASID</th>
<th>TTL</th>
<th>VA[55:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
</tbody>
</table>

**ASID, bits [63:48]**

When $\text{HCR}_\text{EL2}.\text{E2H} = 1$:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

Otherwise:

Reserved, RES0.
TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2 instruction

Accesses to this instruction use the following encodings:

TLBI VALE2{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VALE2(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VALE2(X[t]);
TLBI VALE2IS, TLB Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI VALE2IS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime.
- If \( HCR_{EL2}.E2H == 0 \), the entry is from the final level of the translation table walk.
- If \( HCR_{EL2}.E2H == 1 \), one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of translation table walk that matches the specified ASID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VALE2IS is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE2IS input value bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ASID | TTL | VA[55:12] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**ASID, bits [63:48]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
### TTL to Meaning

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

### VA[55:12], bits [43:0]

Bits [55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits [55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

### Executing the TLBI VALE2IS instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI VALE2IS}\{, <Xt>\}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x10);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VALE2IS(X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        TLBI_VALE2IS(X[t]);
**TLBI VALE2OS, TLB Invalidate by VA, Last level, EL2, Outer Shareable**

The TLBI VALE2OS characteristics are:

**Purpose**

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime.
- If HCR_EL2.E2H == 0, the entry is from the final level of the translation table walk.
- If HCR_EL2.E2H == 1, one of the following applies:
  - The entry is a global entry from the final level of the translation table walk.
  - The entry is a non-global entry from the final level of translation table walk that matches the specified ASID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VALE2OS are UNDEFINED.

**Attributes**

TLBI VALE2OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE2OS input value bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| ASID | TTL | VA[55:12] |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**ASID, bits [63:48]**

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, but only 8 bits are being used in the context being invalidated, the upper bits are RES0.
Otherwise:

Reserved, RES0.

**TTL, bits [47:44]**

*When ARMv8.4-TTL is implemented:*

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.

<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VALE2OS instruction**

Accesses to this instruction use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_VALE2OS(X[t]);
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    UNDEFINED;
  else
    TLBI_VALE2OS(X[t]);
The TLBI VALE3 characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBI VALE3 is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE3 input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47-44</td>
<td>TTL, bits</td>
</tr>
<tr>
<td>55-12</td>
<td>VA[55:12]</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is RES0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VALE3 instruction**

Accesses to this instruction use the following encodings:

```
TLBI VALE3{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_VALE3(X[t]);
The TLBI VALE3IS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

• The entry is a stage 1 translation table entry, from the final level of the translation table walk.
• The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

Attributes

TLBI VALE3IS is a 64-bit System instruction.

Field descriptions

The TLBI VALE3IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>TTL</td>
</tr>
<tr>
<td>61</td>
<td>VA[55:12]</td>
</tr>
<tr>
<td>31</td>
<td>VA[55:12]</td>
</tr>
</tbody>
</table>

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
<table>
<thead>
<tr>
<th>TTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00xx</td>
<td>No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL&lt;1:0&gt; is res0.</td>
</tr>
<tr>
<td>0b01xx</td>
<td>The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
<tr>
<td>0b11xx</td>
<td>The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00: Reserved. Treat as if TTL&lt;3:2&gt; is 0b00. 0b01: Level 1. 0b10: Level 2. 0b11: Level 3.</td>
</tr>
</tbody>
</table>

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, res0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as res0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are res0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are res0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VALE3IS instruction**

Accesses to this instruction use the following encodings:

```
TLBI VALE3IS{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TLBI_VALE3IS(X[t]);

The TLBI VALE3OS characteristics are:

**Purpose**

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VALE3OS are UNDEFINED.

**Attributes**

TLBI VALE3OS is a 64-bit System instruction.

**Field descriptions**

The TLBI VALE3OS input value bit assignments are:

<p>| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|</p>
<table>
<thead>
<tr>
<th>RES0</th>
<th>TTL</th>
<th>VA[55:12]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**TTL, bits [47:44]**

When ARMv8.4-TTL is implemented:

Translation Table Level. Indicates the level of the page table walk that holds the leaf entry for the address being invalidated.
If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

**Otherwise:**

Reserved, RES0.

**VA[55:12], bits [43:0]**

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

**Executing the TLBI VALE3OS instruction**

Accesses to this instruction use the following encodings:

TLBI VALE3OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  TLBI_VALE3OS(X[t]);
The TLBI VMALLE1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of `SCR_EL3.NS`:
  - If `HCR_EL2.{E2H, TGE}` is not `{1, 1}`, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If `HCR_EL2.{E2H, TGE}` is `{1, 1}`, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to the PE that executes this System instruction.

**Note**

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VMALLE1 is a 64-bit System instruction.

**Field descriptions**

TLBI VMALLE1 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI VMALLE1 instruction**

Accesses to this instruction use the following encodings:

```
TLBI VMALLE1{, <Xt>}
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>0b11111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVMALLE1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VMALLE1IS();
  end if;
elsif PSTATE.EL == EL2 then
  TLBI_VMALLE1();
elif PSTATE.EL == EL3 then
  TLBI_VMALLE1();
ellse
  TLBI_VMALLE1();
end if;

27/09/2019 18:22; 6134483bd14dc8c12a99c984c8f3c3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBI VMALLE1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not \{1, 1\}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is \{1, 1\}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VMALLE1IS is a 64-bit System instruction.

**Field descriptions**

TLBI VMALLE1IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI VMALLE1IS instruction**

Accesses to this instruction use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVMALLE1IS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VMALLE1IS();
  end
elsif PSTATE.EL == EL2 then
  TLBI_VMALLE1IS();
elsif PSTATE.EL == EL3 then
  TLBI_VMALLE1IS();
TLBI VMALLE1OS, TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

The TLBI VMALLE1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the Security state described by the current value of SCR_EL3.NS:
  - If HCR_EL2.{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime.
  - If HCR_EL2.{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if SCR_EL3.EEL2==1, then:

- A PE with SCR_EL3.EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==0.
- A PE with SCR_EL3.EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with SCR_EL3.EEL2==1.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

This instruction is present only when ARMv8.4-TLBI is implemented. Otherwise, direct accesses to TLBI VMALLE1OS are UNDEFINED.

**Attributes**

TLBI VMALLE1OS is a 64-bit System instruction.

**Field descriptions**

TLBI VMALLE1OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.
Executing the TLBI VMALLE10S instruction

Accesses to this instruction use the following encodings:

TLBI VMALLE10S{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b11111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB0S == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGITR_EL2.TLBIVMALLE1OS == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TLBI_VMALLE10S();
  endif
elsif PSTATE.EL == EL2 then
  TLBI_VMALLE10S();
elsif PSTATE.EL == EL3 then
  TLBI_VMALLE10S();
The TLBI VMALLS12E1 characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If SCR_EL3.NS is 0, then
  - The entry would be required to translate an address using the Secure EL1&0 translation regime.
  - If ARMv8.4-SecEL2 is implemented and enabled, the entry would be used with the current VMID.
- If SCR_EL3.NS is 1, then:
  - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to the PE that executes this System instruction.

**Note**

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VMALLS12E1 is a 64-bit System instruction.

**Field descriptions**

TLBI VMALLS12E1 ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBI VMALLS12E1 instruction**

Accesses to this instruction use the following encodings:

TLBI VMALLS12E1{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b110</td>
<td>0b11111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x10);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TLBI_VMALLS12E1();
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        TLBI_VMALLE1();
    else
        TLBI_VMALLS12E1();
The TLBI VMALLS12E1IS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.

- If `SCR_EL3.NS` is 0, then
  - The entry would be required to translate an address using the Secure EL1&0 translation regime.
  - If ARMv8.4-SecEL2 is implemented and enabled, the entry would be used with the current VMID.

- If `SCR_EL3.NS` is 1, then:
  - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Note**

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if `SCR_EL3.EEL2==1`, then:

- A PE with `SCR_EL3.EEL2==1` is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with `SCR_EL3.EEL2==0`.
- A PE with `SCR_EL3.EEL2==0` is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with `SCR_EL3.EEL2==1`.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

**Attributes**

TLBI VMALLS12E1IS is a 64-bit System instruction.

**Field descriptions**

TLBI VMALLS12E1IS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.
Executing the TLBI VMALLS12E1IS instruction

Accesses to this instruction use the following encodings:

\[
\text{TLBI VMALLS12E1IS\{, <Xt>\}}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b110</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_VMALLS12E1IS();
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    TLBI_VMALLE1IS();
  else
    TLBI_VMALLS12E1IS();
TLBI VMALLS12E1OS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

The TLBI VMALLS12E1OS characteristics are:

**Purpose**

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If $\text{SCR}_{\text{EL3}}.\text{NS}$ is 0, then
  - The entry would be required to translate an address using the Secure EL1&0 translation regime.
  - If \text{ARMv8.4-SecEL2} is implemented and enabled, the entry would be used with the current VMID.
- If $\text{SCR}_{\text{EL3}}.\text{NS}$ is 1, then:
  - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
  - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

**Note**

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if $\text{SCR}_{\text{EL3}}.\text{EEL2}==1$, then:

- A PE with $\text{SCR}_{\text{EL3}}.\text{EEL2}==1$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $\text{SCR}_{\text{EL3}}.\text{EEL2}==0$.
- A PE with $\text{SCR}_{\text{EL3}}.\text{EEL2}==0$ is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with $\text{SCR}_{\text{EL3}}.\text{EEL2}==1$.
- A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.

For the EL1&0 translation regimes, the invalidation applies to both global entries, and non-global entries with any ASID.

**Configuration**

This instruction is present only when \text{ARMv8.4-TLBI} is implemented. Otherwise, direct accesses to TLBI VMALLS12E1OS are **UNDEFINED**.

**Attributes**

TLBI VMALLS12E1OS is a 64-bit System instruction.

**Field descriptions**

TLBI VMALLS12E1OS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.
Executing the TLBI VMALLS12E1OS instruction

Accesses to this instruction use the following encodings:

TLBI VMALLS12E1OS{, <Xt>}

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b110</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBI_VMALLS12E1OS();
elsif PSTATE.EL == EL3 then
  if !EL2Enabled() then
    TLBI_VMALLE10S();
  else
    TLBI_VMALLS12E1OS();

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TPIDR_EL0, EL0 Read/Write Software Thread ID Register

The TPIDR_EL0 characteristics are:

**Purpose**

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch64 System register TPIDR_EL0 bits [31:0] are architecturally mapped to AArch32 System register TPIDRURW[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TPIDR_EL0 is a 64-bit register.

**Field descriptions**

The TPIDR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Thread ID. Thread identifying information stored by software running at this Exception level.

This field resets to an architecturally UNKNOWN value.

**Accessing the TPIDR_EL0**

Accesses to this register use the following encodings:

MRS <Xt>, TPIDR_EL0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '1' &
  SCR_EL3.FGTEn == '1' && HFGWTR_EL2.TPIDR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TPIDR_EL0;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
  SCR_EL3.FGTEn == '1') &&
  HFGWTR_EL2.TPIDR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TPIDR_EL0;
elsif PSTATE.EL == EL2 then
  return TPIDR_EL0;
elsif PSTATE.EL == EL3 then
  return TPIDR_EL0;

MSR TPIDR_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '1' &
  SCR_EL3.FGTEn == '1' && HFGWTR_EL2.TPIDR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TPIDR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) ||
  SCR_EL3.FGTEn == '1') &&
  HFGWTR_EL2.TPIDR_EL0 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TPIDR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
  TPIDR_EL0 = X[t];
eelsif PSTATE.EL == EL3 then
  TPIDR_EL0 = X[t];
TPIDR_EL1, EL1 Software Thread ID Register

The TPIDR_EL1 characteristics are:

**Purpose**

Provides a location where software executing at EL1 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch64 System register TPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TPIDRPRW[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TPIDR_EL1 is a 64-bit register.

**Field descriptions**

The TPIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Thread ID</td>
</tr>
<tr>
<td>62</td>
<td>Thread ID</td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level. This field resets to an architecturally UNKNOWN value.

**Accessing the TPIDR_EL1**

Accesses to this register use the following encodings:

```
MRS <Xt>, TPIDR_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDR_EL1;
    elsif PSTATE.EL == EL2 then
        return TPIDR_EL1;
    elsif PSTATE.EL == EL3 then
        return TPIDR_EL1;
    end if
end if

MSR TPIDR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        TPIDR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        TPIDR_EL1 = X[t];
    end if
end if
TPIDR_EL2, EL2 Software Thread ID Register

The TPIDR_EL2 characteristics are:

**Purpose**

Provides a location where software executing at EL2 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch64 System register TPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTPIDR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TPIDR_EL2 is a 64-bit register.

**Field descriptions**

The TPIDR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Thread ID</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Thread ID. Thread identifying information stored by software running at this Exception level.

This field resets to an architecturally UNKNOWN value.

**Accessing the TPIDR_EL2**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x090];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    return TPIDR_EL2;
elsif PSTATE.EL == EL3 then
    return TPIDR_EL2;
endfun

MSR TPIDR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x090] = X[t];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    TPIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TPIDR_EL2 = X[t];
endfun
TPIDR_EL3, EL3 Software Thread ID Register

The TPIDR_EL3 characteristics are:

**Purpose**

Provides a location where software executing at EL3 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to TPIDR_EL3 are **UNDEFINED**.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

TPIDR_EL3 is a 64-bit register.

**Field descriptions**

The TPIDR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Thread ID. Thread identifying information stored by software running at this Exception level.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the TPIDR_EL3**

Accesses to this register use the following encodings:

MRS <Xt>, TPIDR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elseif PSTATE.EL == EL3 then
  return TPIDR_EL3;
MSR TPIDR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TPIDR_EL3 = X[t];
TPIDRRO_EL0, EL0 Read-Only Software Thread ID Register

The TPIDRRO_EL0 characteristics are:

**Purpose**

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch64 System register TPIDRRO_EL0 bits [31:0] are architecturally mapped to AArch32 System register TPIDRURO[31:0].

**Attributes**

TPIDRRO_EL0 is a 64-bit register.

**Field descriptions**

The TPIDRRO_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>Thread ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00</td>
<td>11 01 01</td>
</tr>
<tr>
<td>00 00 01</td>
<td>11 01 00</td>
</tr>
<tr>
<td>00 00 02</td>
<td>11 01 11</td>
</tr>
<tr>
<td>00 00 03</td>
<td>11 01 10</td>
</tr>
<tr>
<td>00 00 04</td>
<td>11 00 01</td>
</tr>
<tr>
<td>00 00 05</td>
<td>11 00 11</td>
</tr>
<tr>
<td>00 00 06</td>
<td>11 00 10</td>
</tr>
<tr>
<td>00 00 07</td>
<td>11 00 00</td>
</tr>
</tbody>
</table>

**Accessing the TPIDRRO_EL0**

Accesses to this register use the following encodings:

```
MRS <Xt>, TPIDRRO_EL0
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '1' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDRRO_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDRRO_EL0;
elsif PSTATE.EL == EL2 then
    return TPIDRRO_EL0;
elsif PSTATE.EL == EL3 then
    return TPIDRRO_EL0;

MSR TPIDRRO_EL0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>b11</td>
<td>b011</td>
<td>b1101</td>
<td>b0000</td>
<td>b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDRRO_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    TPIDRRO_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    TPIDRRO_EL0 = X[t];
**TRBBASER_EL1, Trace Buffer Base Address Register**

The TRBBASER_EL1 characteristics are:

**Purpose**

Defines the base address for the trace buffer.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBBASER_EL1 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TRBBASER_EL1 is a 64-bit register.

**Field descriptions**

The TRBBASER_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**BASE**, bits [63:12]

Trace Buffer Base pointer address. (TRBBASER_EL1.BASE << 12) is the address of the first byte in the trace buffer. Bits [11:0] of the Base pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBBASER_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value \( \log_2 \) (smallest implemented translation granule).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [11:0]**

Reserved, RES0.

**Accessing the TRBBASER_EL1**

The PE might ignore a direct write to TRBBASER_EL1 if TRBLIMITR_EL1.E == 1.

Accesses to this register use the following encodings:

```
MRS <Xt>, TRBBASER_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     HDGWR_EL2.TRBBASER_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRBBASER_EL1;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRBBASER_EL1;
  elsif PSTATE.EL == EL3 then
    return TRBBASER_EL1;

MSR TRBBASER_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
     HDGWR_EL2.TRBBASER_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRBBASER_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRBBASER_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    TRBBASER_EL1 = X[t];
TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

**Purpose**

Describes constraints on using the Trace Buffer Extension to software, including whether the Trace Buffer Extension can be programmed at the current Exception level.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBIDR_EL1 are **UNDEFINED**.

**Attributes**

TRBIDR_EL1 is a 64-bit register.

**Field descriptions**

The TRBIDR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>61</td>
<td>F, Flag Updates. Defines whether the address translation performed by the Trace Buffer Extension manages the Access Flag and dirty state.</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>31</td>
<td>F, bit [5]</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, res0.</td>
</tr>
</tbody>
</table>

**Bits [63:6]**

Reserved, res0.

**F, bit [5]**

Flag Updates. Defines whether the address translation performed by the Trace Buffer Extension manages the Access Flag and dirty state.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Trace buffer address translation does not manage the Access flag and dirty state in translation tables.</td>
</tr>
<tr>
<td>1</td>
<td>Trace buffer address translation manages the Access Flag and dirty state in the same way as the MMU on this PE.</td>
</tr>
</tbody>
</table>

**P, bit [4]**

Programming not allowed. The trace buffer is owned by a higher Exception level or by the other Security state.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The owning Exception level is the current Exception level or a lower Exception level, and the owning Security state is the current Security state.</td>
</tr>
<tr>
<td>1</td>
<td>The owning Exception level is a higher Exception level, or the owning Security state is not the current Security state.</td>
</tr>
</tbody>
</table>

The value read from this field depends on the current Exception level and the values of MDCR_EL3.NSTB and MDCR_EL2.E2TB:

- If EL3 is implemented and either MDCR_EL3.NSTB == 0b00 or MDCR_EL3.NSTB == 0b01, meaning the owning Security state is Secure state, this bit reads as one from:
Non-secure EL2.

Non-secure EL1.

If Secure EL2 is implemented and enabled, and MDCR_EL2.E2TB == 0b00, Secure EL1.

- If EL3 is implemented and either MDCR_EL3.NSTB == 0b10 or MDCR_EL3.NSTB == 0b11, meaning the owning Security state is Non-secure state, this bit reads as one from:
  - Secure EL1.
  - If Secure EL2 is implemented, Secure EL2.
  - If EL2 is implemented and MDCR_EL2.E2TB == 0b00, Non-secure EL1.

- If EL3 is not implemented, EL2 is implemented, and MDCR_EL2.E2TB == 0b00, this bit reads as one from EL1.

Otherwise, this bit reads as zero.

**Align, bits [3:0]**

Defines the minimum alignment constraint for writes to TRBPTR_EL1 and TRBTRG_EL1.

<table>
<thead>
<tr>
<th>Align</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Byte.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Halfword.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Word.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Doubleword.</td>
</tr>
<tr>
<td>0b0100</td>
<td>16 bytes.</td>
</tr>
<tr>
<td>0b0101</td>
<td>32 bytes.</td>
</tr>
<tr>
<td>0b0110</td>
<td>64 bytes.</td>
</tr>
<tr>
<td>0b0111</td>
<td>128 bytes.</td>
</tr>
<tr>
<td>0b1000</td>
<td>256 bytes.</td>
</tr>
<tr>
<td>0b1001</td>
<td>512 bytes.</td>
</tr>
<tr>
<td>0b1010</td>
<td>1KB.</td>
</tr>
<tr>
<td>0b1011</td>
<td>2KB.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the TRBIDR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, TRBIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBDT_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRBDT_EL1;
elsif PSTATE.EL == EL2 then
  return TRBDT_EL1;
elsif PSTATE.EL == EL3 then
  return TRBDT_EL1;
TRBLIMITR_EL1, Trace Buffer Limit Address Register

The TRBLIMITR_EL1 characteristics are:

**Purpose**

Defines the top address for the trace buffer, and controls the trace buffer modes and enable.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBLIMITR_EL1 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TRBLIMITR_EL1 is a 64-bit register.

**Field descriptions**

The TRBLIMITR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>LIMIT</th>
<th>RES0</th>
<th>nVM</th>
<th>TM</th>
<th>FM</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LIMIT, bits [63:12]**

Trace Buffer Limit pointer address. (TRBLIMITR_EL1.LIMIT << 12) is the address of the last byte in the trace buffer plus one. Bits [11:0] of the Limit pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBLIMITR_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log2(smallest implemented translation granule).

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [11:6]**

Reserved, RES0.

**nVM, bit [5]**

Address mode.

<table>
<thead>
<tr>
<th>nVM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace buffer pointers are virtual addresses.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace buffer pointers are:</td>
</tr>
<tr>
<td></td>
<td>• Physical address in the owning security state if the owning translation regime has no stage 2 translation.</td>
</tr>
<tr>
<td></td>
<td>• Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally UNKNOWN value.
**TM, bits [4:3]**

Trigger mode.

<table>
<thead>
<tr>
<th>TM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Stop on trigger. Flush then stop collection and raise maintenance interrupt on Trigger Event.</td>
</tr>
<tr>
<td>0b01</td>
<td>IRQ on trigger. Continue collection and raise maintenance interrupt on Trigger Event.</td>
</tr>
<tr>
<td>0b11</td>
<td>Ignore trigger. Continue collection and do not raise maintenance interrupt on Trigger Event.</td>
</tr>
</tbody>
</table>

All other values are reserved.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**FM, bits [2:1]**

Trace buffer mode.

<table>
<thead>
<tr>
<th>FM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Fill mode. Stop collection and raise maintenance interrupt on current write pointer wrap.</td>
</tr>
<tr>
<td>0b01</td>
<td>Wrap mode. Continue collection and raise maintenance interrupt on current write pointer wrap.</td>
</tr>
<tr>
<td>0b11</td>
<td>Circular Buffer mode. Continue collection and do not raise maintenance interrupt on current write pointer wrap.</td>
</tr>
</tbody>
</table>

All other values are reserved.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**E, bit [0]**

Trace Buffer Extension enable.

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace Buffer Extension disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace Buffer Extension enabled by this control.</td>
</tr>
</tbody>
</table>

Regardless of the value of this bit, the Trace Buffer Extension is disabled when `SelfHostedTraceEnabled() == FALSE`. All output is discarded by the Trace Buffer Extension when it is disabled.

This field resets to 0.

**Accessing the TRBLIMITR_EL1**

The PE might ignore a direct write to TRBLIMITR_EL1, other than a direct write that modifies TRBLIMITR_EL1.E, if TRBLIMITR_EL1.E == 1.

Accesses to this register use the following encodings:

```
MRS <Xt>, TRBLIMITR_EL1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
  UNDEFINED;
elsif PSTATE_EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRBLIMITR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRBLIMIT_EL1;
endif
elsif PSTATE_EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRBLIMIT_EL1 = X[t];
endif
elsif PSTATE_EL == EL3 then
  TRBLIMIT_EL1 = X[t];
else
  TRBLIMIT_EL1 = X[t];
endif
TRBMAR_EL1, Trace Buffer Memory Attribute Register

The TRBMAR_EL1 characteristics are:

**Purpose**

Controls Trace Buffer Extension accesses to memory.

If the trace buffer pointers specify virtual addresses, the address properties are defined by the translation tables and this register is ignored.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBMAR_EL1 are undefined.

RW fields in this register reset to architecturally unknown values.

**Attributes**

TRBMAR_EL1 is a 64-bit register.

**Field descriptions**

The TRBMAR_EL1 bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

<table>
<thead>
<tr>
<th>RES0</th>
<th>SH</th>
<th>Attr[7:4]</th>
<th>Attr[3:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:10]**

Reserved, RES0.

**SH, bits [9:8]**

Trace buffer shareability domain. Defines the shareability domain for Normal memory used by the trace buffer.

<table>
<thead>
<tr>
<th>SH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not shared.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner shareable.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field resets to an architecturally unknown value.

**Attr[7:4], bits [7:4]**

Trace buffer memory type and attributes. Defines the memory type and, for Normal memory, the Outer cacheability attributes, for memory addressed by the trace buffer.
### TRBMAR_EL1, Trace Buffer Memory Attribute Register

<table>
<thead>
<tr>
<th>Attr[7:4]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device memory. The device memory type is defined by TRBMAR_EL1.Attr[3:0].</td>
</tr>
<tr>
<td>0b0001</td>
<td>Normal memory, Outer write-through cacheable, transient Write allocate.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Normal memory, Outer write-through cacheable, transient Read allocate.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Normal memory, Outer write-through cacheable, transient Read and Write allocate.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Outer non-cacheable.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Normal memory, Outer write-back cacheable, transient Write allocate.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Normal memory, Outer write-back cacheable, transient Read allocate.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Normal memory, Outer write-back cacheable, transient Read and Write allocate.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Normal memory, Outer write-through cacheable, No allocate.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Normal memory, Outer write-through cacheable, Write allocate.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Normal memory, Outer write-through cacheable, Read allocate.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Normal memory, Outer write-through cacheable, Read and Write allocate.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Normal memory, Outer write-back cacheable, No allocate.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Normal memory, Outer write-back cacheable, Write allocate.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Normal memory, Outer write-back cacheable, Read allocate.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Normal memory, Outer write-back cacheable, Read and Write allocate.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

### Attr[3:0], bits [3:0]

When TRBMAR_EL1.Attr[7:4] == 0b0000:

Trace buffer memory attributes. Defines the Device memory attributes for memory addressed by the trace buffer.

<table>
<thead>
<tr>
<th>Attr[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Device-nGnRE.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Device-nGRE.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Device-GRE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field resets to an architecturally UNKNOWN value.

When TRBMAR_EL1.Attr[7:4] != 0b0000:

Trace buffer memory attributes. Defines the Inner cacheability attributes for memory addressed by the trace buffer.
<table>
<thead>
<tr>
<th>Attr[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Normal memory, Inner write-through cacheable, transient Write allocate.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Normal memory, Inner write-through cacheable, transient Read allocate.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Normal memory, Inner write-through cacheable, transient Read and Write allocate.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Inner non-cacheable.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Normal memory, Inner write-back cacheable, transient Write allocate.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Normal memory, Inner write-back cacheable, transient Read allocate.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Normal memory, Inner write-back cacheable, transient Read and Write allocate.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Normal memory, Inner write-through cacheable, No allocate.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Normal memory, Inner write-through cacheable, Write allocate.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Normal memory, Inner write-through cacheable, Read allocate.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Normal memory, Inner write-through cacheable, Read and Write allocate.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Normal memory, Inner write-back cacheable, No allocate.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Normal memory, Inner write-back cacheable, Write allocate.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Normal memory, Inner write-back cacheable, Read allocate.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Normal memory, Inner write-back cacheable, Read and Write allocate.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Accessing the TRBMAR_EL1**

The PE might ignore a direct write to TRBMAR_EL1 if TRBLIMITR_EL1.E == 1.

Accesses to this register use the following encodings:

MRS <Xt>, TRBMAR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRBMAR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS != '0' && MDCR_EL3.NSTB != '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRBMAR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRBMAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  TRBMAR_EL1 = X[t];

MSR TRBMAR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.TRBMAR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS != '0' && MDCR_EL3.NSTB != '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRBMAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRBMAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  TRBMAR_EL1 = X[t];
The TRBPTR_EL1 characteristics are:

**Purpose**

Defines the current write pointer for the trace buffer.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBPTR_EL1 are **UNDEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

TRBPTR_EL1 is a 64-bit register.

**Field descriptions**

The TRBPTR_EL1 bit assignments are:

<p>|   | 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |</p>
<table>
<thead>
<tr>
<th></th>
<th>PTR</th>
<th>PTR</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
</tr>
</tbody>
</table>

**PTR, bits [63:0]**

Trace Buffer current write pointer address.

Defines the virtual address of the next entry to be written to the trace buffer.

The architecture places restrictions on the values that software can write to the pointer.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the TRBPTR_EL1**

The PE might ignore a direct write to TRBPTR_EL1 if **TRBLIMITR_EL1**.E == 1.

Accesses to this register use the following encodings:

**MRS <Xt>, TRBPTR_EL1**

<table>
<thead>
<tr>
<th></th>
<th>0b11</th>
<th>0b000</th>
<th>0b1001</th>
<th>0b1011</th>
<th>0b001</th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRBPTR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
  MDCR_EL3.NSTB != '01' then
  AArch64.SystemAccessTrap(EL3, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
  MDCR_EL3.NSTB != '11' then
  AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRBPTR_EL1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
    MDCR_EL3.NSTB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
  MDCR_EL3.NSTB != '11' then
  AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRBPTR_EL1;
elsif PSTATE.EL == EL3 then
  return TRBPTR_EL1;

MSR TRBPTR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.TRBPTR_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
  AArch64.SystemAccessTrap(EL2, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
  MDCR_EL3.NSTB != '01' then
  AArch64.SystemAccessTrap(EL3, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
  MDCR_EL3.NSTB != '11' then
  AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRBPTR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' &&
    MDCR_EL3.NSTB != '01' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' &&
  MDCR_EL3.NSTB != '11' then
  AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRBPTR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  TRBPTR_EL1 = X[t];
The TRBSR_EL1 characteristics are:

**Purpose**

Provides syndrome information to software for a trace buffer management event.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBSR_EL1 are undefined. RW fields in this register reset to architecturally unknown values.

**Attributes**

TRBSR_EL1 is a 64-bit register.

**Field descriptions**

The TRBSR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>EC (Event class)</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>IRQ</td>
</tr>
<tr>
<td>59</td>
<td>WRAP</td>
</tr>
<tr>
<td>58</td>
<td>RES0</td>
</tr>
<tr>
<td>57</td>
<td>S</td>
</tr>
<tr>
<td>56</td>
<td>RES0</td>
</tr>
<tr>
<td>55</td>
<td>MSS</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**EC, bits [31:26]**

Event class. Top-level description of the cause of the trace buffer management event.

<table>
<thead>
<tr>
<th>EC</th>
<th>Meaning</th>
<th>MSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b100100</td>
<td>Stage 1 Data Abort on write to trace buffer.</td>
<td>MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer</td>
</tr>
<tr>
<td>0b100101</td>
<td>Stage 2 Data Abort on write to trace buffer.</td>
<td>MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer</td>
</tr>
<tr>
<td>0b000000</td>
<td>Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes.</td>
<td>MSS encoding for other trace buffer management events</td>
</tr>
</tbody>
</table>

All other values are reserved.

On a Cold reset, this field resets to an architecturally unknown value.

**Bits [25:23]**

Reserved, RES0.
**Interrupt Request, bit [22]**

Maintenance interrupt status.

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt is not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt is asserted.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Triggered, bit [21]**

Triggered.

<table>
<thead>
<tr>
<th>TRG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Detected Trigger has been observed since this bit was last cleared to zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Detected Trigger has been observed since this bit was last cleared to zero.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Wrapped, bit [20]**

Wrapped.

<table>
<thead>
<tr>
<th>WRAP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The current write pointer has not wrapped since this bit was last cleared to zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>The current write pointer has wrapped since this bit was last cleared to zero.</td>
</tr>
</tbody>
</table>

For each byte of trace the Trace Buffer Extension Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this bit is set to 1.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Reserved, bit [19]**

Reserved, **RES0**.

**External Abort, bit [18]**

External Abort.

<table>
<thead>
<tr>
<th>EA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An External Abort has not been asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>An External Abort has been asserted and detected by the Trace Buffer Extension.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if the PE never sets this bit as the result of an External Abort.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Stopped, bit [17]**

Stopped.

<table>
<thead>
<tr>
<th>S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Collection has not been stopped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Collection is stopped.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
Bit [16]
Reserved, RES0.

**MSS, bits [15:0]**

Management Event Specific Syndrome. Contains syndrome specific to the management event.
The syndrome contents for each management event are described in the following sections.
On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

### MSS encoding for other trace buffer management events

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>RES0</td>
</tr>
<tr>
<td>14</td>
<td>BSC</td>
</tr>
</tbody>
</table>

**Bits [15:6]**
Reserved, RES0.

**BSC, bits [5:0]**
Trace buffer status code.

<table>
<thead>
<tr>
<th>BSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Collection not stopped.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information.</td>
</tr>
</tbody>
</table>

All other values are reserved.

### MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>RES0</td>
</tr>
<tr>
<td>14</td>
<td>FSC</td>
</tr>
</tbody>
</table>

**Bits [15:6]**
Reserved, RES0.

**FSC, bits [5:0]**
Fault status code.
## TRBSR_EL1, Trace Buffer Status/syndrome Register

### FSC Meaning

<table>
<thead>
<tr>
<th>FSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address Size fault, level 0.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address Size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address Size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address Size fault, level 3.</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 0.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access Flag fault, level 0.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access Flag fault, level 1.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access Flag fault, level 2.</td>
</tr>
<tr>
<td>0b001011</td>
<td>Access Flag fault, level 3.</td>
</tr>
<tr>
<td>0b001100</td>
<td>Permission fault, level 0.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External Abort on write.</td>
</tr>
<tr>
<td>0b010001</td>
<td>Asynchronous External Abort on write.</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External Abort on translation table walk or hardware update of translation table, level 0.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External Abort on translation table walk or hardware update of translation table, level 1.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External Abort on translation table walk or hardware update of translation table, level 2.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External Abort on translation table walk or hardware update of translation table, level 3.</td>
</tr>
<tr>
<td>0b010001</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b100000</td>
<td>TLB Conflict fault.</td>
</tr>
<tr>
<td>0b110001</td>
<td>Unsupported atomic hardware update fault.</td>
</tr>
</tbody>
</table>

All other values are reserved.

### Accessing the TRBSR_EL1

The PE might ignore a direct write to TRBSR_EL1 if TRBLIMITR_EL1.E == 1.

Accesses to this register use the following encodings:

**MRS <Xt>, TRBSR_EL1**

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>op0</td>
<td>0b11</td>
<td>op1</td>
<td>0b000</td>
<td>CRn</td>
<td>0b1001</td>
</tr>
</tbody>
</table>

---

Page 1758
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && FDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && FDCR_EL3.NSTB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && FDCR_EL3.NSTB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRBSR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && FDCR_EL3.NSTB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && FDCR_EL3.NSTB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBSR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        TRBSR_EL1 = X[t];

MSR TRBSR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.TRBSR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && FDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && FDCR_EL3.NSTB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && FDCR_EL3.NSTB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRBSR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && FDCR_EL3.NSTB != '01' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && FDCR_EL3.NSTB != '11' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBSR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        TRBSR_EL1 = X[t];
TRBTRG_EL1, Trace Buffer Trigger Counter Register

The TRBTRG_EL1 characteristics are:

**Purpose**

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

**Configuration**

This register is present only when TRBE is implemented. Otherwise, direct accesses to TRBTRG_EL1 are UNDEFINED. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TRBTRG_EL1 is a 64-bit register.

**Field descriptions**

The TRBTRG_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31-0</td>
<td>Trigger count (TRG).</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**TRG, bits [31:0]**

Trigger count.

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

TRBTRG_EL1 decrements by 1 for every byte of trace written to the trace buffer when all of the following are true:

- TRBTRG_EL1 is nonzero.
- TRBSR_EL1.TRG is set to 1.

The architecture places restrictions on the values that software can write to the counter.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRBTRG_EL1**

The PE might ignore a direct write to TRBTRG_EL1 if TRBLIMITR_EL1.E == 1.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWR_EL2.TRBTRG_EL1 == 'I' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRBTRG_EL1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' && MDCR_EL3.NSTB != '01' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.NS == '1' && MDCR_EL3.NSTB != '11' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRBTRG_EL1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    TRBTRG_EL1 = X[t];
else
    TRBTRG_EL1 = X[t];
end

MSR TRBTRG_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b110</td>
</tr>
</tbody>
</table>
TRCACATR<n>, Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

**Purpose**

Defines the type of access for the corresponding TRCACVR<n> Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

**Configuration**

AArch64 System register TRCACATR<n> bits [63:0] are architecturally mapped to External register TRCACATR<n>[63:0].

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACATR<n> are UNDEFINED.

**Attributes**

TRCACATR<n> is a 64-bit register.

**Field descriptions**

The TRCACATR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit Location</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>EXLEVEL_NS_EL2, bit [14]</td>
</tr>
<tr>
<td></td>
<td>When Non-secure EL2 is implemented:</td>
</tr>
<tr>
<td></td>
<td>Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0b0</td>
</tr>
<tr>
<td></td>
<td>0b1</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
EXLEVEL_NS_EL1, bit [13]

When Non-secure EL1 is implemented:
Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Non-secure EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Non-secure EL1.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

EXLEVEL_NS_EL0, bit [12]

When Non-secure EL0 is implemented:
Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Non-secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Non-secure EL0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

EXLEVEL_S_EL3, bit [11]

When HaveEL(EL3):
Secure EL3 address comparison control. Controls whether a comparison can occur at EL3 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL3.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

EXLEVEL_S_EL2, bit [10]

When HaveEL(EL2) and ARMv8.4-SecEL2 is implemented:
Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.
EXLEVEL_S_EL2

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL2.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [9]

When Secure EL1 is implemented:

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL1.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [8]

When Secure EL0 is implemented:

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

CONTEXT, bits [6:4]

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:
The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

If TRCIDR4.UMCIDC == 0b0000 and TRCIDR4.VMVMIDC == 0b0000, then this field is RES0.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**CONTEXTTYPE, bits [3:2]**

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons:

<table>
<thead>
<tr>
<th>CONTEXTTYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b01</td>
<td>The Address Comparator is dependent on the Context Identifier Comparator that the CONTEXT field specifies. If both the Context Identifier Comparator and the address comparison match, the Address Comparator signals a match.</td>
</tr>
<tr>
<td>0b10</td>
<td>The Address Comparator is dependent on the Virtual Context Identifier Comparator that the CONTEXT field specifies. If both the Virtual Context Identifier Comparator and the address comparison match, the Address Comparator signals a match.</td>
</tr>
<tr>
<td>0b11</td>
<td>The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that the CONTEXT field specifies. If the Context Identifier Comparator, the Virtual Context Identifier Comparator and address comparison all match, the Address Comparator signals a match.</td>
</tr>
</tbody>
</table>

If TRCIDR4.UMCIDC == 0b0000, then bit [2] is RES0.

If TRCIDR4.VMVMIDC == 0b0000, then bit [3] is RES0.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [1:0]**

Reserved, RES0.

**Accessing the TRCACATR<n>**

Must be programmed if any of the following are true:

- TRCBBCtlr.RANGE[n/2] == 0b1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 0b1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 0b1.
- TRCVIIECTLR.EXCLUDE[n/2] == 0b1.
- TRCVIIECTLR.INCLUDE[n/2] == 0b1.
• TRCVISSCTLR.START[n] == 0b1.
• TRCVISSCTLR.STOP[n] == 0b1.
• TRCSSCCR<>.ARC[n/2] == 0b1.
• TRCSSCCR<>.SAC[n] == 0b1.
• TRCQCTLR.RANGE[n/2] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCACATR<n>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0010</td>
<td>n[2:0]:0b0</td>
<td>0b01:n[3]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
if CPACR_EL1.TTA == '1' then
AArch64.SystemAccessTrap(EL1, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
AArch64.SystemAccessTrap(EL3, 0x18);
else
return TRCACATR[UInt(op2<0>:CRm<3:1>)];
elsif PSTATE.EL == EL2 then
if CPTR_EL2.TTA == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
AArch64.SystemAccessTrap(EL3, 0x18);
else
return TRCACATR[UInt(op2<0>:CRm<3:1>)];
elsif PSTATE.EL == EL3 then
if CPTR_EL3.TTA == '1' then
AArch64.SystemAccessTrap(EL3, 0x18);
else
return TRCACATR[UInt(op2<0>:CRm<3:1>)];

MSR TRCACATR<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0010</td>
<td>n[2:0]:0b0</td>
<td>0b01:n[3]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCACATR[UInt(op2<0>:CRm<3:1>)] = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCACATR[UInt(op2<0>:CRm<3:1>)] = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCACATR[UInt(op2<0>:CRm<3:1>)] = X[t];
The TRCACVR<n> characteristics are:

**Purpose**

Contains the address value.

**Configuration**

AArch64 System register TRCACVR<n> bits [63:0] are architecturally mapped to External register TRCACVR<n>[63:0].

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACVR<n> are UNDEFINED.

**Attributes**

TRCACVR<n> is a 64-bit register.

**Field descriptions**

The TRCACVR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>62</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>61</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>60</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>59</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>58</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>57</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>56</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>55</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>54</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>53</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>52</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>51</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>50</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>49</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>48</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>47</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>46</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>45</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>44</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>43</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>42</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>41</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>40</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>39</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>38</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>37</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>36</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>35</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>34</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>33</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>32</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>31</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>30</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>29</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>28</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>27</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>26</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>25</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>24</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>23</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>22</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>21</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>20</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>19</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>18</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>17</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>16</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>15</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>14</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>13</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>12</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>11</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>10</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>9</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>8</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>7</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>6</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>5</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>4</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>3</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>2</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>1</td>
<td>ADDRESS</td>
</tr>
<tr>
<td>0</td>
<td>ADDRESS</td>
</tr>
</tbody>
</table>

**ADDRESS, bits [63:0]**

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in the TRCACVR<n>. This requires that the trace analyzer always programs all implemented bits of the TRCACVR<n>.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as the virtual address size supported by the PE.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCACVR<n>**

Must be programmed if any of the following are true:

- TRCBBCTLR.RANGE[n/2] == 0b1.
- TRCRSCTLR<a>.GROUP == 0b0100 and TRCRSCTLR<a>.SAC[n] == 0b1.
- TRCRSCTLR<a>.GROUP == 0b0101 and TRCRSCTLR<a>.ARC[n/2] == 0b1.
TRCVIIECTLR.EXCLUDE[n/2] == 0b1.
TRCVIIECTLR.INCLUDE[n/2] == 0b1.
TRCVISSCTLR.START[n] == 0b1.
TRCVISSCTLR.STOP[n] == 0b1.
TRCSSCCR<=.ARC[n/2] == 0b1.
TRCSSCCR<=.SAC[n] == 0b1.
TRCOCTL.RANGE[n/2] == 0b1.

Writes are CONSTRANED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCACVR<n>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0010</td>
<td>n[2:0]:0b0</td>
<td>0b00:n[3]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsf PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
    return TRCACVR[UInt(op2<0>:CRm<3:1>)];
elsf PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
    return TRCACVR[UInt(op2<0>:CRm<3:1>)];
elsf PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
    return TRCACVR[UInt(op2<0>:CRm<3:1>)];

MSR TRCACVR<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0010</td>
<td>n[2:0]:0b0</td>
<td>0b00:n[3]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCACVR[UInt(op2<0>:CRm<3:1>)] = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCACVR[UInt(op2<0>:CRm<3:1>)] = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCACVR[UInt(op2<0>:CRm<3:1>)] = X[t];
    endif
The TRCAUTHSTATUS characteristics are:

**Purpose**

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information see the CoreSight Architecture Specification.

**Configuration**

AArch64 System register TRCAUTHSTATUS bits [31:0] are architecturally mapped to External register TRCAUTHSTATUS[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are UNDEFINED.

**Attributes**

TRCAUTHSTATUS is a 64-bit register.

**Field descriptions**

The TRCAUTHSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61-59</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td>SNID, bits [7:6] (Secure Non-invasive Debug)</td>
</tr>
<tr>
<td>57-50</td>
<td>Reserved</td>
</tr>
<tr>
<td>49-47</td>
<td>Reserved</td>
</tr>
<tr>
<td>46-45</td>
<td>Reserved</td>
</tr>
<tr>
<td>44-43</td>
<td>Reserved</td>
</tr>
<tr>
<td>42-41</td>
<td>Reserved</td>
</tr>
<tr>
<td>40-39</td>
<td>Reserved</td>
</tr>
<tr>
<td>38-37</td>
<td>Reserved</td>
</tr>
<tr>
<td>36-35</td>
<td>Reserved</td>
</tr>
<tr>
<td>34-33</td>
<td>Reserved</td>
</tr>
<tr>
<td>32-31</td>
<td>Bits [63:8] (Reserved, RES0)</td>
</tr>
<tr>
<td>30-29</td>
<td>Reserved</td>
</tr>
<tr>
<td>28-26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25-24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23-22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21-20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19-18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17-14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13-10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9-8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7-0</td>
<td>Bits [63:8] (Reserved, RES0)</td>
</tr>
</tbody>
</table>

**SNID, bits [7:6]**

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

<table>
<thead>
<tr>
<th>SNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Secure non-invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When EL3 is implemented, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b00.

When EL3 is not implemented and the PE is Secure, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

**SID, bits [5:4]**

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.
<table>
<thead>
<tr>
<th>SID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Secure invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

**NSID, bits [3:2]**

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-secure non-invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When EL3 is implemented, this field reads as 0b11.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b11.

When EL3 is not implemented and the PE is Secure, this field reads as 0b00.

**NSID, bits [1:0]**

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-secure invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

**Accessing the TRCAUTHSTATUS**

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

Accesses to this register use the following encodings:

```
MRS <Xt>, TRCAUTHSTATUS
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRAUTHSTATUS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCAUTHSTATUS;
    elsif PSTATE_EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCAUTHSTATUS;
        elsif PSTATE_EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCAUTHSTATUS;
            end if
        end if
    end if
end if
TRCAUXCTLR, Auxillary Control Register

The TRCAUXCTLR characteristics are:

Purpose

The function of this register is IMPLEMENTATION DEFINED.

Configuration

AArch64 System register TRCAUXCTLR bits [31:0] are architecturally mapped to External register TRCAUXCTLR[31-0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCAUXCTLR are UNDEFINED.

Attributes

TRCAUXCTLR is a 64-bit register.

Field descriptions

The TRCAUXCTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

On a Trace unit reset, this field resets to 0.

Accessing the TRCAUXCTLR

If this register is set to nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

Accesses to this register use the following encodings:

MRS <Xt>, TRCAUXCTLR

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRAUXCTRLR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCAUXCTRLR;
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCAUXCTRLR;
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCAUXCTRLR;
else
  TRCAUXCTRLR = X[t];
endif

MSR TRCAUXCTRLR, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRAUXCTRLR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRCAUXCTRLR = X[t];
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRCAUXCTRLR = X[t];
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRCAUXCTRLR = X[t];
TRCBBCTLR, Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

Purpose

Controls the regions in the memory map where branch broadcasting is active.

Configuration

AArch64 System register TRCBBCTRLR bits [31:0] are architecturally mapped to External register TRCBBCTRLR[31:0].

This register is present only when ETE is implemented, TRCIDR0.TRCBB == 0b1 and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCBBCTRLR are UNDEFINED.

Attributes

TRCBBCTRLR is a 64-bit register.

Field descriptions

The TRCBBCTRLR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 | MODE | RANGE<m>, bit [m] |

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Mode.

<table>
<thead>
<tr>
<th>MODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by RANGE. If RANGE == 0x00 then branch broadcasting is active for all instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Include Mode. Branch broadcasting is active for instructions in the address ranges defined by RANGE. If RANGE == 0x00 then the behavior of the trace unit is CONSTRAINED UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE<m>, bit [m], for m = 0 to 7

Address range field.

Selects which Address Range Comparators are in use with branch broadcasting.
| RANGE<
  m< | Meaning |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address range that Address Range Comparator m defines, is not selected.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address range that Address Range Comparator m defines, is selected.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

### Accessing the TRCBBCTLR

Must be programmed if TRCCONFIGR.BB == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

\[
\text{MRS } <\text{Xt}>, \text{ TRCBBCTLR}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCBBCTLR;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCBBCTLR;
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCBBCTLR;
            end if
        else
            return TRCBBCTLR;
        end if
    end if
\]

\[
\text{MSR } \text{TRCBBCTLR, } <\text{Xt}>
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1111</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRCBBCTLR = X[t];
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  TRCBBCTLR = X[t];
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCBBCTLR = X[t];
TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR characteristics are:

**Purpose**

Set the threshold value for cycle counting.

**Configuration**

AArch64 System register TRCCCCTLR bits [31:0] are architecturally mapped to External register TRCCCCTLR[31:0].

This register is present only when ETE is implemented and TRCIDR0.TRCCCI == 0b1. Otherwise, direct accesses to TRCCCCTLR are UNDEFINED.

**Attributes**

TRCCCCTLR is a 64-bit register.

**Field descriptions**

The TRCCCCTLR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | THRESHOLD |

**Bits [63:12]**

Reserved, RES0.

**THRESHOLD, bits [11:0]**

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in TRCIDR3.CCITMIN. If the THRESHOLD value is smaller than the value in TRCIDR3.CCITMIN then the behavior is CONSTRAINED UNPREDICTABLE. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when TRCCONFIGR.CCI is set to enable instruction trace cycle counting, results in CONSTRAINED UNPREDICTABLE behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCCCCTLR**

Must be programmed if TRCCONFIGR.CCI == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCCCCTLR

| op0 | op1 | CRn | CRm | op2 |
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCCCTLR;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCCCTLR;
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                TRCCCCTLR = X[t];
        else
            TRCCCCTLR = X[t];
    else
        TRCCCCTLR = X[t];

MSR TRCCCCTLR, <Xt>
TRCCIDCCTRLR0, Context Identifier Comparator Control Register 0

The TRCCIDCCTRLR0 characteristics are:

**Purpose**

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 0 to 3.

**Configuration**

AArch64 System register TRCCIDCCTRLR0 bits [31:0] are architecturally mapped to External register TRCCIDCCTRLR0[31:0].

This register is present only when ETE is implemented, TRCIDR4.NUMCIDC > 0x0 and TRCIDR2.CIDSIZE > 0b00000. Otherwise, direct accesses to TRCCIDCCTRLR0 are UNDEFINED.

**Attributes**

TRCCIDCCTRLR0 is a 64-bit register.

**Field descriptions**

The TRCCIDCCTRLR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>COMP3&lt;m&gt;, bit [m+24]</td>
</tr>
<tr>
<td>61</td>
<td>COMP2&lt;m&gt;, bit [m+16]</td>
</tr>
<tr>
<td>60</td>
<td>COMP1&lt;m&gt;, bit [m+8]</td>
</tr>
<tr>
<td>59</td>
<td>COMP0&lt;m&gt;, bit [m]</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**COMP3<m>, bit [m+24], for m = 0 to 7**

When TRCIDR4.NUMCIDC > 3:

TRCCIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR3. Each bit in this field corresponds to a byte in TRCCIDCVR3.

<table>
<thead>
<tr>
<th>COMP3&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**COMP2<m>, bit [m+16], for m = 0 to 7**
When TRCIDR4.NUMCIDC > 2:

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.

<table>
<thead>
<tr>
<th>COMP2&lt;\texttt{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>The trace unit includes TRCCIDCVR2((m\times8+7):(m\times8)) when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>The trace unit ignores TRCCIDCVR2((m\times8+7):(m\times8)) when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \(m \geq \text{TRCIDR2.CIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

\textbf{COMP1<\texttt{m}>}, bit \([m+8]\), for \(m = 0\) to 7

When TRCIDR4.NUMCIDC > 1:

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

<table>
<thead>
<tr>
<th>COMP1&lt;\texttt{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>The trace unit includes TRCCIDCVR1((m\times8+7):(m\times8)) when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>The trace unit ignores TRCCIDCVR1((m\times8+7):(m\times8)) when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \(m \geq \text{TRCIDR2.CIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

\textbf{COMP0<\texttt{m}>}, bit \([m]\), for \(m = 0\) to 7

When TRCIDR4.NUMCIDC > 0:

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

<table>
<thead>
<tr>
<th>COMP0&lt;\texttt{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>The trace unit includes TRCCIDCVR0((m\times8+7):(m\times8)) when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>The trace unit ignores TRCCIDCVR0((m\times8+7):(m\times8)) when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \(m \geq \text{TRCIDR2.CIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
Accessing the TRCCIDCCTLR0

If software uses the TRCCIDCVR<n> registers, for n = 0 to 3, then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCCIDCVR<n> to 0x00.

If any bit is 0b1 and the relevant byte in TRCCIDCVR<n> is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCCIDCCTLR0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCCIDCCTLR0;
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCCTLR0;
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCCTLR0;

MSR TRCCIDCCTLR0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

TRCCIDCCTLR0, Context Identifier Comparator Control Register 0
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTeN == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTRL0 = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTRL0 = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTRL0 = X[t];
TRCIDCCTRLR1, Context Identifier Comparator Control Register 1

The TRCIDCCTRLR1 characteristics are:

**Purpose**

Contains Context identifier mask values for the TRCIDCVR<n> registers, for n = 4 to 7.

**Configuration**

AArch64 System register TRCIDCCTRLR1 bits [31:0] are architecturally mapped to External register TRCIDCCTRLR1[31:0].

This register is present only when ETE is implemented, TRCIDR4.NUMCIDC > 0x4 and TRCIDR2.CIDSIZE > 0b00000. Otherwise, direct accesses to TRCIDCCTRLR1 are UNDEFINED.

**Attributes**

TRCIDCCTRLR1 is a 64-bit register.

**Field descriptions**

The TRCIDCCTRLR1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    | RES0 | COMP7<m>, bit [m+24] | COMP6<m>, bit [m+16] | COMP5<m>, bit [m+8] | COMP4<m>, bit [m] |

**Bits [63:32]**

Reserved, RES0.

COMP7<m>, bit [m+24], for m = 0 to 7

When TRCIDR4.NUMCIDC > 7:

TRCIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCIDCVR7. Each bit in this field corresponds to a byte in TRCIDCVR7.

<table>
<thead>
<tr>
<th>COMP7&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m ≥ TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP6<m>, bit [m+16], for m = 0 to 7
When TRCIDR4.NUMCIDC > 6:

TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.

<table>
<thead>
<tr>
<th>COMP6&lt;6&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP5<5>, bit [m+8], for m = 0 to 7

When TRCIDR4.NUMCIDC > 5:

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

<table>
<thead>
<tr>
<th>COMP5&lt;5&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP4<4>, bit [m], for m = 0 to 7

When TRCIDR4.NUMCIDC > 4:

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

<table>
<thead>
<tr>
<th>COMP4&lt;4&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
Accessing the TRCCIDCCTLR1

If software uses the TRCCIDCVR\<n> registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCCIDCVR\<n> to 0x00.

If any bit is 0b1 and the relevant byte in TRCCIDCVR\<n> is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCCIDCCTLR1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCCTLR1;
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCCTLR1;
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCCTLR1;

MSR TRCCIDCCTLR1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    end
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTLR1 = X[t];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTLR1 = X[t];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTLR1 = X[t];
end
TRCCIDCVR<n>, Context Identifier Comparator Value Registers <n>, n = 0 - 7

The TRCCIDCVR<n> characteristics are:

**Purpose**

Contains a Context identifier value.

**Configuration**

AArch64 System register TRCCIDCVR<n> bits [63:0] are architecturally mapped to External register TRCCIDCVR<n>[63:0].

This register is present only when ETE is implemented and TRCIDR4.NUMCIDC > n. Otherwise, direct accesses to TRCCIDCVR<n> are UNDEFINED.

**Attributes**

TRCCIDCVR<n> is a 64-bit register.

**Field descriptions**

The TRCCIDCVR<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| VALUE | VALUE |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**VALUE, bits [63:0]**

Context identifier value. The width of this field is indicated by TRCIDR2.CIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCCIDCVR<n>**

Must be programmed if any of the following are true:

- TRCRSCLTR<a>.GROUP == 0b0110 and TRCRSCLTR<a>.CID[n] == 0b1.
- TRCACATR<a>.CONTEXTTYPE == 0b01 or 0b11 and TRCACATR<a>.CONTEXT == n.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>n[2:0]:0b0</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCVR[UInt(CRm<3:1>)];
  end if;
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCVR[UInt(CRm<3:1>)];
  end if;
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCIDCVR[UInt(CRm<3:1>)];
  end if;

MSR TRCCIDCVR<n>, Context Identifier Comparator Value Registers <n>, n = 0 - 7

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>n[2:0]:0b0</td>
<td>0b000</td>
</tr>
</tbody>
</table>
TRCCLAIMCLR, Claim Tag Clear Register

The TRCCLAIMCLR characteristics are:

**Purpose**

In conjunction with TRCCLAIMSET, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information see the CoreSight Architecture Specification.

**Configuration**

AArch64 System register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register TRCCLAIMCLR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCLAIMCLR are UNDEFINED.

**Attributes**

TRCCLAIMCLR is a 64-bit register.

**Field descriptions**

The TRCCLAIMCLR bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLR&lt;m&gt;</td>
<td>Claim Tag Clear. Indicates the current status of the Claim Tag bit m, and is used to clear Claim Tag bit m to 0b0.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CLR&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>On a read: Claim Tag bit m is not set. On a write: Ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>On a read: Claim Tag bit m is set. On a write: Clear Claim tag bit m to 0b0.</td>
</tr>
</tbody>
</table>

The number of Claim Tag bits implemented is indicated in TRCCLAIMSET.

On a Trace unit reset, this field resets to 0.

**Accessing the TRCCLAIMCLR**

Accesses to this register use the following encodings:

MRS <Xt>, TRCCLAIMCLR

| op0 | op1 | CRn | CRm | op2 |
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRCCLAIM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCLAIMCLR;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCCLAIMCLR;
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCCLAIMCLR;
  end

MSR TRCCLAIMCLR, <xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRCCLAIM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCCLAIMCLR = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCCLAIMCLR = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCCLAIMCLR = X[t];
TRCCLAIMSET, Claim Tag Set Register

The TRCCLAIMSET characteristics are:

Purpose

In conjunction with TRCCLAIMCLR, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCCLAIMSET bits [31:0] are architecturally mapped to External register TRCCLAIMSET[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCLAIMSET are UNDEFINED.

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as 0b1111.

Attributes

TRCCLAIMSET is a 64-bit register.

Field descriptions

The TRCCLAIMSET bit assignments are:

<table>
<thead>
<tr>
<th>m</th>
<th>SET&lt;m&gt;</th>
<th>bit [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

SET<m>, bit [m], for m = 0 to 31

Claim Tag Set. Indicates whether Claim Tag bit m is implemented, and is used to set Claim Tag bit m to 0b1.

<table>
<thead>
<tr>
<th>SET&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>On a read: Claim Tag bit m is not implemented.</td>
</tr>
<tr>
<td></td>
<td>On a write: Ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>On a read: Claim Tag bit m is implemented.</td>
</tr>
<tr>
<td></td>
<td>On a write: Set Claim Tag bit m to 0b1.</td>
</tr>
</tbody>
</table>

Accessing the TRCCLAIMSET

Accesses to this register use the following encodings:

MRS <Xt>, TRCCLAIMSET

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & ((!HaveEL(EL3) || SCR_EL3.FGTEn == '1') & HDFGRTR_EL2.TRCCLAIM == '1') then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCLAIMSET;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCCLAIMSET;
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCCLAIMSET;
  end if
end if

MSR TRCCLAIMSET, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & ((!HaveEL(EL3) || SCR_EL3.FGTEn == '1') & HDFGWTR_EL2.TRCCLAIM == '1') then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCCLAIMSET = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCCLAIMSET = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCCLAIMSET = X[t];
TRCCNTCTRLR\(<n>\), Counter Control Register \(<n>\), n = 0 - 3

The TRCCNTCTRLR\(<n>\) characteristics are:

**Purpose**

Controls the operation of Counter \(<n>\).

**Configuration**

AArch64 System register TRCCNTCTRLR\(<n>\) bits [31:0] are architecturally mapped to External register TRCCNTCTRLR\(<n>\[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTCTRLR\(<n>\) are **UNDEFINED**.

**Attributes**

TRCCNTCTRLR\(<n>\) is a 64-bit register.

**Field descriptions**

The TRCCNTCTRLR\(<n>\) bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 - 18</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>CNTCHAIN</td>
<td>For TRCCNTCTRLR3 and TRCCNTCTRLR1, this bit controls whether the Counter decrements when a reload event occurs for Counter (&lt;n-1&gt;).</td>
</tr>
<tr>
<td>16</td>
<td>RLDSELF</td>
<td>Controls whether a reload event occurs for the Counter, when the Counter reaches zero.</td>
</tr>
</tbody>
</table>

CNTCHAIN is not implemented for TRCCNTCTRLR0 and TRCCNTCTRLR2.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

RLDSELF, bit [16]

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.
On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**RLDEVENT_TYPE, bit [15]**

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes a reload event for Counter <n>.

<table>
<thead>
<tr>
<th>RLDEVENT_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector: TRCCNTCTRL&lt;n&gt;.RLDEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCCNTCTRL&lt;n&gt;.RLDEVENT.SEL[3:0] selects the Resource Selector pair; from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTRL&lt;n&gt;.RLDEVENT.SEL[4] is RES.0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [14:13]**

Reserved, RES.0.

**RLDEVENT_SEL, bits [12:8]**

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTRL<n>.RLDEVENT.TYPE controls whether TRCCNTCTRL<n>.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes a reload event for Counter <n>.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**CNTEVENT_TYPE, bit [7]**

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes Counter <n> to decrement.

<table>
<thead>
<tr>
<th>CNTEVENT_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector: TRCCNTCTRL&lt;n&gt;.CNTEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCCNTCTRL&lt;n&gt;.CNTEVENT.SEL[3:0] selects the Resource Selector pair; from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTRL&lt;n&gt;.CNTEVENT.SEL[4] is RES.0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTR<n>.CNTEVENT.TYPE controls whether TRCCNTCTR<n>.CNTEVENT_SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes Counter <n> to decrement.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCCNTCTR<n>

Must be programmed if TRCRSCTR<a>.GROUP == 0b0010 and TRCRSCTR<a>.COUNTERS[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCCNTCTR<n>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b01:n[1:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCCNTCTR[UInt(CRm<1:0>)];
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCNTCTR[UInt(CRm<1:0>)];
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCNTCTR[UInt(CRm<1:0>)];
  endif
else
  return TRCCNTCTR[UInt(CRm<1:0>)];
endif

MSR TRCCNTCTR<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b01:n[1:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TRCCNTCTLR[UInt(CRm<1:0>)] = X[t];
    end if
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCNTCTLR[UInt(CRm<1:0>)] = X[t];
    end if
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCNTCTLR[UInt(CRm<1:0>)] = X[t];
    end if
TRCCNTRLDVR\(n\), Counter Reload Value Register \(n\), \(n = 0 - 3\)

The TRCCNTRLDVR\(n\) characteristics are:

**Purpose**

This sets or returns the reload count value for Counter \(n\).

**Configuration**

AArch64 System register TRCCNTRLDVR\(n\) bits \([31:0]\) are architecturally mapped to External register TRCCNTRLDVR\(n\)[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMCNTR > \(n\). Otherwise, direct accesses to TRCCNTRLDVR\(n\) are **UNDEFINED**.

**Attributes**

TRCCNTRLDVR\(n\) is a 64-bit register.

**Field descriptions**

The TRCCNTRLDVR\(n\) bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-16</td>
<td>Reserved, RESO</td>
<td>0</td>
</tr>
<tr>
<td>15-0</td>
<td>VALUE</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RESO.

**VALUE, bits [15:0]**

Contains the reload value for Counter \(n\). When a reload event occurs for Counter \(n\) then the trace unit copies the VALUE\(n\) field into Counter \(n\).

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the TRCCNTRLDVR\(n\)**

Must be programmed if TRCRSCTL\(a\).GROUP == 0b0010 and TRCRSCTL\(a\).COUNTERS[\(n\)] == 0b1.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b00:n[1:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCNTRLDVR[UInt(CRm<1:0>)];
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTRLDVR[UInt(CRm<1:0>)];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTRLDVR[UInt(CRm<1:0>)];

MSR TRCCNTRLDVR<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b00:n[1:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCNTRLDVR[UInt(CRm<1:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTRLDVR[UInt(CRm<1:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTRLDVR[UInt(CRm<1:0>)] = X[t];

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TRCCNTVR<n>, Counter Value Register <n>, n = 0 - 3

The TRCCNTVR<n> characteristics are:

**Purpose**

This sets or returns the value of Counter <n>.

**Configuration**

AArch64 System register TRCCNTVR<n> bits [31:0] are architecturally mapped to External register TRCCNTVR<n>[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTVR<n> are UNDEFINED.

**Attributes**

TRCCNTVR<n> is a 64-bit register.

**Field descriptions**

The TRCCNTVR<n> bit assignments are:

```
  63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32
  31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
```

**RES0**

Reserved, RES0.

**VALUE**, bits [15:0]

Contains the count value of Counter.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCCNTVR<n>**

Must be programmed if TRCRSCTRLR<a>.GROUP == 0b0010 and TRCRSCTRLR<a>.COUNTERS[n] == 0b1.

 Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

 Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings:

```
   MRS <Xt>, TRCCNTVR<n>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b10:01[0:1]</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELusingAAarch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELusingAAarch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCCNTVRn == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCCNTVR[UInt(CRm<1:0>)];
end if;
elsif PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELusingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELusingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCCNTVRn == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCCNTVR[UInt(CRm<1:0>)];
end if;
TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

**Purpose**

Controls the tracing options.

**Configuration**

AArch64 System register TRCCONFIGR bits [31:0] are architecturally mapped to External register TRCCONFIGR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCONFIGR are UNDEFINED.

**Attributes**

TRCCONFIGR is a 64-bit register.

**Field descriptions**

The TRCCONFIGR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RES0.

**Bit [15]**

When TRCIDR2.VMIDOPT == 0b00:

Reserved, RES0.

Virtual context identifier selection control.

VTTBR_EL2. VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b10:

Reserved, RES1.

Virtual context identifier selection control.

CONTEXTIDR_EL2. PROCID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b01:

Virtual context identifier selection control.
**VMIDOPT**

<table>
<thead>
<tr>
<th>VMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>VTTBR_EL2: VMID is used as the Virtual context identifier.</td>
</tr>
<tr>
<td>0b1</td>
<td>CONTEXTIDR_EL2: PROCID is used as the Virtual context identifier.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**QE, bits [14:13]**

When TRCIDR0.QSUPP == 0b01:

Q element generation control.

<table>
<thead>
<tr>
<th>QE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q elements are disabled.</td>
</tr>
<tr>
<td>0b01</td>
<td>Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When TRCIDR0.QSUPP == 0b10:

Q element generation control.

<table>
<thead>
<tr>
<th>QE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q elements are disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Q elements with instruction counts are enabled. Q elements without instruction counts are enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When TRCIDR0.QSUPP == 0b11:

Q element generation control.

<table>
<thead>
<tr>
<th>QE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q elements are disabled.</td>
</tr>
<tr>
<td>0b01</td>
<td>Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Q elements with instruction counts are enabled. Q elements without instruction counts are enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**RS, bit [12]**

When TRCIDR0.RETSTACK == 0b1:

Return stack control.

<table>
<thead>
<tr>
<th>RS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Return stack is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Return stack is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RES0.

**TS, bit [11]**
When TRCIDR0.TSSIZE != 0b00000:

Global timestamp tracing control.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Global timestamp tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Global timestamp tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**Bits [10:8]**
Reserved, RES0.

**VMID, bit [7]**
When TRCIDR2.VMIDSIZE != 0b00000:

Virtual context identifier tracing control.

<table>
<thead>
<tr>
<th>VMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual context identifier tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual context identifier tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**CID, bit [6]**
When TRCIDR2.CIDSIZE != 0b00000:

Context identifier tracing control.

<table>
<thead>
<tr>
<th>CID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Context identifier tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Context identifier tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**Bit [5]**
Reserved, RES0.
**CCI, bit [4]**

When TRCIDR0.TRCCCI == 0b1:

Cycle counting instruction tracing control.

<table>
<thead>
<tr>
<th>CCI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting instruction tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting instruction tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**BB, bit [3]**

When TRCIDR0.TRCBB == 0b1:

Branch broadcasting control.

<table>
<thead>
<tr>
<th>BB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Branch broadcasting is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Branch broadcasting is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**Bits [2:1]**

Reserved, RES0.

**Bit [0]**

Reserved, RES1.

**Accessing the TRCCONFIGR**

Must always be programmed.

TRCCONFIGR.QE must be set to 0b00 if TRCCONFIGR.BB is not 0b0.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

**MRS <Xt>, TRCCONFIGR**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCCONFIGR;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCCONFIGR;
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCCONFIGR;
  end
end

MSR TRCCONFIGR, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b00</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCCONFIGR = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCCONFIGR = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCCONFIGR = X[t];
The TRCDEVARCH characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

AArch64 System register TRCDEVARCH bits [31:0] are architecturally mapped to External register TRCDEVARCH[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVARCH are UNDEFINED.

**Attributes**

TRCDEVARCH is a 64-bit register.

**Field descriptions**

The TRCDEVARCH bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ARCHITECT | PRESENT | REVISION | ARCHVER | ARCHPART |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**ARCHITECT, bits [31:21]**

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

<table>
<thead>
<tr>
<th>ARCHITECT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01000111011</td>
<td>JEP106 continuation code 0x4, ID code 0x3B. Arm Limited.</td>
</tr>
</tbody>
</table>

Other values are defined by the JEDEC JEP106 standard.

This field reads as 0x23B.

**PRESENT, bit [20]**

DEVARCH Present. Defines that the DEVARCH register is present.

<table>
<thead>
<tr>
<th>PRESENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Device Architecture information not present.</td>
</tr>
<tr>
<td>0b1</td>
<td>Device Architecture information present.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.
REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

<table>
<thead>
<tr>
<th>REVISION</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ETE Version 1.0.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

<table>
<thead>
<tr>
<th>ARCHVER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0101</td>
<td>ETE Version 1.</td>
</tr>
</tbody>
</table>

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12]. This field reads as 0x5.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

<table>
<thead>
<tr>
<th>ARCHPART</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0xA13    | Arm PE trace architecture.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0]. This field reads as 0xA13.

Accessing the TRCDEVARCH

Accesses to this register use the following encodings:

MRS <Xt>, TRCDEVARCH

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1111</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCDEVARCH;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCDEVARCH;
  end
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCDEVARCH;
end
TRCDEVID, Device Configuration Register

The TRCDEVID characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

AArch64 System register TRCDEVID bits [31:0] are architecturally mapped to External register TRCDEVID[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVID are UNDEFINED.

**Attributes**

TRCDEVID is a 64-bit register.

**Field descriptions**

The TRCDEVID bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>Reserved</td>
</tr>
<tr>
<td>55</td>
<td>Reserved</td>
</tr>
<tr>
<td>54</td>
<td>Reserved</td>
</tr>
<tr>
<td>53</td>
<td>Reserved</td>
</tr>
<tr>
<td>52</td>
<td>Reserved</td>
</tr>
<tr>
<td>51</td>
<td>Reserved</td>
</tr>
<tr>
<td>50</td>
<td>Reserved</td>
</tr>
<tr>
<td>49</td>
<td>Reserved</td>
</tr>
<tr>
<td>48</td>
<td>Reserved</td>
</tr>
<tr>
<td>47</td>
<td>Reserved</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Accessing the TRCDEVID**

Accesses to this register use the following encodings:

MRS <Xt>, TRCDEVID

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsifHaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCDEVID;
  end if;
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCDEVID;
  end if;
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCDEVID;
  end if;
else
  return TRCDEVID;
end if;
**TRCEVENTCTL0R, Event Control 0 Register**

The TRCEVENTCTL0R characteristics are:

### Purpose

Controls the generation of ETEEvents.

### Configuration

AArch64 System register TRCEVENTCTL0R bits [31:0] are architecturally mapped to External register TRCEVENTCTL0R[31:0].

This register is present only when ETE is implemented and TRCIDR4.NUMRSPAIR != 0b0000. Otherwise, direct accesses to TRCEVENTCTL0R are **UNDEFINED**.

### Attributes

TRCEVENTCTL0R is a 64-bit register.

### Field descriptions

The TRCEVENTCTL0R bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| Res0 | EVENT3_TYPE | Res0 | EVENT3_SEL | EVENT2_TYPE | RES0 | EVENT2_SEL | EVENT1_TYPE | RES0 | EVENT1_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   |

**Bits [63:32]**

Reserved, Res0.

**EVENT3_TYPE, bit [31]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT3_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT3_SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3_SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3_SEL[4] is Res0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, Res0.
Bits [30:29]

Reserved, RES0.

**EVENT3_SEL, bits [28:24]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[3] == 0b1, then Event element 3 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**EVENT2_TYPE, bit [23]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT2_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:21]

Reserved, RES0.

**EVENT2_SEL, bits [20:16]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.
When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[2] == 0b1, then Event element 2 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT1_TYPE, bit [15]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT1_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [14:13]

Reserved, RES0.

EVENT1_SEL, bits [12:8]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[1] == 0b1, then Event element 1 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT0_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Chooses the type of Resource Selector.
<table>
<thead>
<tr>
<th>EVENT0_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT0_SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
</tbody>
</table>
| 0b1        | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0_SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event.

TRCEVENTCTL0R.EVENT0_SEL[4] is RES0.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT0_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0_SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[0] == 0b1, then Event element 0 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing the TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCEVENTCTL0R

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEVENTCTL0R;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCEVENTCTL0R;
    elseif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCEVENTCTL0R;
    end
end

MSR TRCEVENTCTL0R, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCEVENTCTL0R = X[t];
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEVENTCTL0R = X[t];
    elseif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEVENTCTL0R = X[t];

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TRCEVENTCTL1R, Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

Purpose

Controls the behavior of the ETEEvents that TRCEVENTCTRL0R selects.

Configuration

AArch64 System register TRCEVENTCTL1R bits [31:0] are architecturally mapped to External register TRCEVENTCTL1R[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are UNDEFINED.

Attributes

TRCEVENTCTL1R is a 64-bit register.

Field descriptions

The TRCEVENTCTL1R bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>LPOVERRIDE, bit [12]</td>
</tr>
<tr>
<td>11</td>
<td>ATB, bit [11]</td>
</tr>
</tbody>
</table>

LPOVERRIDE, bit [12]

When TRCIDR5.LPOVERRIDE == 0b1:

Low-power Override Mode select.

<table>
<thead>
<tr>
<th>LPOVERRIDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

ATB, bit [11]

When TRCIDR5.ATBTRIG == 0b1:

AMBA Trace Bus (ATB) trigger enable.
If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:

- \( \text{ATID} = \text{0x7D} \).
- \( \text{ATDATA} \) to the value of \text{TRCTRACEIDR}.

If the width of ATDATA is greater than the width of \text{TRCTRACEIDR}.TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

<table>
<thead>
<tr>
<th>ATB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ATB trigger is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>ATB trigger is enabled.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**Bits [10:4]**

Reserved, RES0.

**INSTEN\(<m>\), bit \([m]\), for \(m = 0\) to \(3\)**

Event element control.

<table>
<thead>
<tr>
<th>INSTEN(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit does not generate an Event element (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit generates an Event element (m).</td>
</tr>
</tbody>
</table>

This bit is RES0 if \(m \geq \) the number indicated by \text{TRCIDR0}.NUMEVENT.

**Accessing the TRCEVENTCTL1R**

Must be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>MRS &lt;Xt&gt;, TRCEVENTCTL1R</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{op0}</td>
</tr>
<tr>
<td>0b10</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCEVENTCTL1R;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCEVENTCTL1R = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCEVENTCTL1R = X[t];
end if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCEVENTCTL1R;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCEVENTCTL1R = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCEVENTCTL1R = X[t];
end

MSR TRCEVENTCTL1R, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b001</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCEVENTCTL1R = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCEVENTCTL1R = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCEVENTCTL1R = X[t];
end

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TRCEXTINSELR<n> characteristics are:

**Purpose**

Use this to set, or read, which External Inputs are resources to the trace unit.

**Configuration**

AArch64 System register TRCEXTINSELR<n> bits [31:0] are architecturally mapped to External register TRCEXTINSELR<n>[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMEXTINSEL > n. Otherwise, direct accesses to TRCEXTINSELR<n> are UNDEFINED.

**Attributes**

TRCEXTINSELR<n> is a 64-bit register.

**Field descriptions**

The TRCEXTINSELR<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | evtCount |

**Bits [63:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range 0x0000 to 0x003F are common architectural and microarchitectural events.
- PMU event numbers in the range 0x0040 to 0x00FF are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range 0x00C0 to 0x03FF are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range 0x0000 to 0x003F, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.
Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCEXTINSELR<n>**

Must be programmed if any of the following is true: TRCRSCTRLR<a>.GROUP == 0b0000 and TRCRSCTRLR<a>.EXTIN[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

**MRS <Xt>, TRCEXTINSELR<n>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b10:n[1:0]</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCEXTINSELR[UInt(CRm<1:0>)];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEXTINSELR[UInt(CRm<1:0>)];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEXTINSELR[UInt(CRm<1:0>)];
    endif
else
    return TRCEXTINSELR[UInt(CRm<1:0>)];
endif

**MSR TRCEXTINSELR<n>, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b10:n[1:0]</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TRCEXTINSELR[UInt(CRm<1:0>)] = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCEXTINSELR[UInt(CRm<1:0>)] = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCEXTINSELR[UInt(CRm<1:0>)] = X[t];
    endif
The TRCIDR0 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR0 bits [31:0] are architecturally mapped to External register TRCIDR0[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR0 are undefined.

**Attributes**

TRCIDR0 is a 64-bit register.

**Field descriptions**

The TRCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>COMMTRANS</td>
</tr>
<tr>
<td>61</td>
<td>COMMOPT</td>
</tr>
<tr>
<td>60</td>
<td>TSSIZE</td>
</tr>
<tr>
<td>59</td>
<td>RES0</td>
</tr>
<tr>
<td>58</td>
<td>TRCEXDATA</td>
</tr>
<tr>
<td>57</td>
<td>QSUPPQFILT</td>
</tr>
<tr>
<td>56</td>
<td>CONDTYPE</td>
</tr>
<tr>
<td>55</td>
<td>NUMEVENT</td>
</tr>
<tr>
<td>54</td>
<td>RETSTACK</td>
</tr>
<tr>
<td>53</td>
<td>TRCCCI</td>
</tr>
<tr>
<td>52</td>
<td>TRCOND</td>
</tr>
<tr>
<td>51</td>
<td>TRCBB</td>
</tr>
<tr>
<td>50</td>
<td>TRCDATA</td>
</tr>
<tr>
<td>49</td>
<td>INSTP0</td>
</tr>
<tr>
<td>48</td>
<td>RES1</td>
</tr>
<tr>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
</tr>
<tr>
<td>43</td>
<td>42</td>
</tr>
<tr>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
</tr>
<tr>
<td>38</td>
<td>37</td>
</tr>
<tr>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>36</td>
<td>35</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>34</td>
<td>33</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>COMMTRANS</td>
</tr>
<tr>
<td>29</td>
<td>COMMOPT</td>
</tr>
<tr>
<td>28</td>
<td>TSSIZE</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>TRCEXDATA</td>
</tr>
<tr>
<td>25</td>
<td>QSUPPQFILT</td>
</tr>
<tr>
<td>24</td>
<td>CONDTYPE</td>
</tr>
<tr>
<td>23</td>
<td>NUMEVENT</td>
</tr>
<tr>
<td>22</td>
<td>RETSTACK</td>
</tr>
<tr>
<td>21</td>
<td>TRCCCI</td>
</tr>
<tr>
<td>20</td>
<td>TRCOND</td>
</tr>
<tr>
<td>19</td>
<td>TRCBB</td>
</tr>
<tr>
<td>18</td>
<td>TRCDATA</td>
</tr>
<tr>
<td>17</td>
<td>INSTP0</td>
</tr>
<tr>
<td>16</td>
<td>RES1</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
</tbody>
</table>

**Bits [63:31]**

Reserved, RES0.

**COMMTRANS, bit [30]**

Transaction Start element behavior.

<table>
<thead>
<tr>
<th>COMMTRANS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Transaction Start elements are P0 elements.</td>
</tr>
<tr>
<td>0b1</td>
<td>Transaction Start elements are not P0 elements.</td>
</tr>
</tbody>
</table>

**COMMOPT, bit [29]**

Indicates how the commit field in Cycle count packets is interpreted.

<table>
<thead>
<tr>
<th>COMMOPT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Commit mode 0. Cycle count packets includes the Commit element to which the Cycle Count element is attached.</td>
</tr>
<tr>
<td>0b1</td>
<td>Commit mode 1. Cycle count packets do not include Commit elements. The Cycle Count element is attached to the most recent Commit element.</td>
</tr>
</tbody>
</table>

This bit reads-as-one if TRCIDR0.TRCCCI == 0b1 and TRCIDR0.MAXSPEC == 0x0. This bit reads-as-zero if TRCIDR0.TRCCCI == 0b0.
**TSSIZE, bits [28:24]**

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.

<table>
<thead>
<tr>
<th>TSSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Global timestamping not implemented.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Global timestamping implemented with a 64-bit timestamp value.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b01000.

**Bits [23:18]**

Reserved, RES0.

**TRCEXDATA, bit [17]**

*When TRCIDR0.TRCDATA != 0b00:*

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>TRCEXDATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing of data transfers for exceptions and exception returns not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing of data transfers for exceptions and exception returns implemented.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**QSUPP, bits [16:15]**

Indicates that the trace unit implements Q element support.

<table>
<thead>
<tr>
<th>QSUPP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Q element support is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Q element support is implemented, and only supports Q elements with instruction counts.</td>
</tr>
<tr>
<td>0b10</td>
<td>Q element support is implemented, and only supports Q elements without instruction counts.</td>
</tr>
<tr>
<td>0b11</td>
<td>Q element support is implemented, and supports:</td>
</tr>
<tr>
<td></td>
<td>• Q elements with instruction counts.</td>
</tr>
<tr>
<td></td>
<td>• Q elements without instruction counts.</td>
</tr>
</tbody>
</table>

**QFILT, bit [14]**

*When TRCIDR0.QSUPP != 0b00:*

Indicates if the trace unit implements Q element filtering.

<table>
<thead>
<tr>
<th>QFILT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Q element filtering is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Q element filtering is implemented.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.
CONDTYPE, bits [13:12]

When TRCIDR0.TRCCOND == 0b1:

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>CONDTYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Conditional instructions are traced with an indication of whether they pass or fail their condition code check.</td>
</tr>
<tr>
<td>0b01</td>
<td>Conditional instructions are traced with an indication of the APSR condition flags.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == 0b0000:

Indicates the number of ETEEvents implemented.

<table>
<thead>
<tr>
<th>NUMEVENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The trace unit supports 0 ETEEvents.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When TRCIDR4.NUMRSPAIR != 0b0000:

Indicates the number of ETEEvents implemented.

<table>
<thead>
<tr>
<th>NUMEVENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The trace unit supports 1 ETEEvent.</td>
</tr>
<tr>
<td>0b01</td>
<td>The trace unit supports 2 ETEEvents.</td>
</tr>
<tr>
<td>0b10</td>
<td>The trace unit supports 3 ETEEvents.</td>
</tr>
<tr>
<td>0b11</td>
<td>The trace unit supports 4 ETEEvents.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

<table>
<thead>
<tr>
<th>RETSTACK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Return stack not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Return stack implemented.</td>
</tr>
</tbody>
</table>

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.
TRCCCI

<table>
<thead>
<tr>
<th>TRCCCI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

TRCOND, bit [6]

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

<table>
<thead>
<tr>
<th>TRCOND</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Conditional instruction tracing not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Conditional instruction tracing implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

TRCBB, bit [5]

Indicates if the trace unit implements branch broadcasting.

<table>
<thead>
<tr>
<th>TRCBB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Branch broadcasting not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Branch broadcasting implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

TRCDATA, bits [4:3]

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

<table>
<thead>
<tr>
<th>TRCDATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Data tracing not implemented.</td>
</tr>
<tr>
<td>0b11</td>
<td>Data tracing implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

INSTP0, bits [2:1]

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

<table>
<thead>
<tr>
<th>INSTP0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Load and store instructions are not P0 instructions.</td>
</tr>
<tr>
<td>0b11</td>
<td>Load and store instructions are P0 instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

Bit [0]

Reserved, RES1.

Accessing the TRCIDR0

Accesses to this register use the following encodings:
MRS <Xt>, TRCIDR0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCIDR0;
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR0;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR0;
  endif
else
  return TRCIDR0;
endif

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TRCIDR1 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR1 bits [31:0] are architecturally mapped to External register `TRCIDR1[31:0]`. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR1 are **UNDEFINED**.

**Attributes**

TRCIDR1 is a 64-bit register.

**Field descriptions**

The TRCIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>62</td>
<td>DESIGNER</td>
<td>Indicates which company designed the trace unit. The permitted values of this field are the same as <code>MIDR_EL1</code> Implementer.</td>
</tr>
<tr>
<td>61</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>60</td>
<td>RES1</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td>TRCARCHMAJ</td>
<td>Major architecture version.</td>
</tr>
<tr>
<td>58</td>
<td>TRCARCHMIN</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td>REVISION</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**DESIGNER, bits [31:24]**

Indicates which company designed the trace unit. The permitted values of this field are the same as `MIDR_EL1` Implementer.

**Bits [23:16]**

Reserved, RES0.

**Bits [15:12]**

Reserved, RES1.

**TRCARCHMAJ, bits [11:8]**

Major architecture version.

<table>
<thead>
<tr>
<th>TRCARCHMAJ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b1111.
**TRCARCHMIN, bits [7:4]**

Minor architecture version.

<table>
<thead>
<tr>
<th>TRCARCHMIN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b1111.

**REVISION, bits [3:0]**

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of:

- The trace registers.
- The OS Lock registers.

Arm recommends that the initial implementation sets REVISION == 0x0 and the field then increments for any subsequent implementations. However, it is acceptable to omit some values or use another scheme to identify the revision number.

Arm recommends that TRCPIDR2.REVISION == TRCIDR1.REVISION. However, in situations where it is difficult to align these fields, such as with a metal layer fix then it is acceptable to change the REVISION fields independently.

**Accessing the TRCIDR1**

Accesses to this register use the following encodings:

```markdown
MRS <Xt>, TRCIDR1
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1001</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
else if PSTATE.EL == EL1 then
  if CPCR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  else
    EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    else
      HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
      else
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
          AArch64.SystemAccessTrap(EL3, 0x18);
        else
          return TRCIDR1;
  endif;
else if PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
      else
        return TRCIDR1;
  endif;
else if PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR1;
else
  return TRCIDR1;
```
TRCIDR10, ID Register 10

The TRCIDR10 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR10 bits [31:0] are architecturally mapped to External register TRCIDR10[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR10 are **UNDEFINED**.

**Attributes**

TRCIDR10 is a 64-bit register.

**Field descriptions**

The TRCIDR10 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | NUMP1KEY |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**NUMP1KEY, bits [31:0]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

**Accessing the TRCIDR10**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR10

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL2) && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
else
    return TRCIDR10;
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return TRCIDR10;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
else
    return TRCIDR10;
TRCIDR11, ID Register 11

The TRCIDR11 characteristics are:

### Purpose

Returns the tracing capabilities of the trace unit.

### Configuration

AArch64 System register TRCIDR11 bits [31:0] are architecturally mapped to External register TRCIDR11[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR11 are UNDEFINED.

### Attributes

TRCIDR11 is a 64-bit register.

### Field descriptions

The TRCIDR11 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>NUMP1SPC</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
</tr>
<tr>
<td>62</td>
<td>61</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**NUMP1SPC, bits [31:0]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

### Accessing the TRCIDR11

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR11

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR11;
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR11;
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR11;
    endif

The TRCIDR12 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR12 bits [31:0] are architecturally mapped to External register TRCIDR12[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR12 are UNDEFINED.

**Attributes**

TRCIDR12 is a 64-bit register.

**Field descriptions**

The TRCIDR12 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>NUMCONDKEY</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**NUMCONDKEY, bits [31:0]**

When TRCIDR0.TRCCOND == 0b1:

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

**Accessing the TRCIDR12**

Accesses to this register use the following encodings:

```
MRS <Xt>, TRCIDR12
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCIDR12;
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR12;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR12;
  endif
The TRCIDR13 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR13 bits [31:0] are architecturally mapped to External register TRCIDR13[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR13 are UNDEFINED.

**Attributes**

TRCIDR13 is a 64-bit register.

**Field descriptions**

The TRCIDR13 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>31</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>30</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>29</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>28</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>27</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>26</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>25</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>24</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>23</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>22</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>21</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>20</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>19</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>18</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>17</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>16</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>15</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>14</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>13</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>12</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>11</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>10</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>9</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>8</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>7</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>6</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>5</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>4</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>3</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>2</td>
<td>NUMCONDSPC, bits [31:0]</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**NUMCONDSPC, bits [31:0]**

When TRCIDR0.TRCCOND == 0b1:

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

**Accessing the TRCIDR13**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR13

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCIDR13;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCIDR13;
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCIDR13;
  else
    return TRCIDR13;
The TRCIDR2 characteristics are:

## Purpose

Returns the tracing capabilities of the trace unit.

## Configuration

AArch64 System register TRCIDR2 bits [31:0] are architecturally mapped to External register TRCIDR2[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR2 are undefined.

## Attributes

TRCIDR2 is a 64-bit register.

## Field descriptions

The TRCIDR2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | WFXMODE | VMIDOPT | CCSIZE | DVSIZE | DASIZE | VMIDSIZE | CIDSIZE | IASIZE |

### Bits [63:32]

Reserved, RES0.

### WFXMODE, bit [31]

Indicates whether WFI and WFE instructions are classified as P0 instructions:

<table>
<thead>
<tr>
<th>WFXMODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>WFI and WFE instructions are not classified as P0 instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>WFI and WFE instructions are classified as P0 instructions.</td>
</tr>
</tbody>
</table>

### VMIDOPT, bits [30:29]

When TRCIDR2.VMIDSIZE != 0b0000:

Indicates the options for Virtual context identifier selection.

<table>
<thead>
<tr>
<th>VMIDOPT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES0.</td>
</tr>
<tr>
<td>0b01</td>
<td>Virtual context identifier selection supported. TRCCONFIGR.VMIDOPT is implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b10.
Reserved, RES0.

**CCSIZE, bits [28:25]**

When TRCIDR0.TRCCCI == 0b1:

Indicates the size of the cycle counter.

<table>
<thead>
<tr>
<th>CCSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The cycle counter is 12 bits in length.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The cycle counter is 13 bits in length.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The cycle counter is 14 bits in length.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The cycle counter is 15 bits in length.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The cycle counter is 16 bits in length.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The cycle counter is 17 bits in length.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The cycle counter is 18 bits in length.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The cycle counter is 19 bits in length.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The cycle counter is 20 bits in length.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Reserved, RES0.

**DVSIZE, bits [24:20]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>DVSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Data value tracing not implemented.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Data value tracing has a maximum of 32-bit data values.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Data value tracing has a maximum of 64-bit data values.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Reserved, RES0.

**DASIZE, bits [19:15]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>DASIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Data address tracing not implemented.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Data address tracing has a maximum of 32-bit data addresses.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Data address tracing has a maximum of 64-bit data addresses.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Reserved, RES0.
VMIDSIZE, bits [14:10]

When HaveEL(EL2):

Indicates the trace unit Virtual context identifier size.

<table>
<thead>
<tr>
<th>VMIDSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Virtual context identifier tracing is not supported.</td>
</tr>
<tr>
<td>0b00001</td>
<td>8-bit Virtual context identifier size.</td>
</tr>
<tr>
<td>0b00010</td>
<td>16-bit Virtual context identifier size.</td>
</tr>
<tr>
<td>0b00100</td>
<td>32-bit Virtual context identifier size.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00100.

Otherwise:

Reserved, RES0.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

<table>
<thead>
<tr>
<th>CIDSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Context identifier tracing is not supported.</td>
</tr>
<tr>
<td>0b00100</td>
<td>32-bit Context identifier size.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00100.

IASIZE, bits [4:0]

Virtual instruction address size.

<table>
<thead>
<tr>
<th>IASIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01000</td>
<td>Maximum of 32-bit instruction address size.</td>
</tr>
<tr>
<td>0b01000</td>
<td>Maximum of 64-bit instruction address size.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b01000.

Accessing the TRCIDR2

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b1010</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCIDR2;
    endelsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR2;
    endelsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR2;
TRCIDR3, ID Register 3

The TRCIDR3 characteristics are:

**Purpose**

Returns the base architecture of the trace unit.

**Configuration**

AArch64 System register TRCIDR3 bits [31:0] are architecturally mapped to External register TRCIDR3[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR3 are UNDEFINED.

**Attributes**

TRCIDR3 is a 64-bit register.

**Field descriptions**

The TRCIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>NOOVERFLOW</td>
</tr>
<tr>
<td>61</td>
<td>NUMPROC[2:0]</td>
</tr>
<tr>
<td>60</td>
<td>SYSTALL</td>
</tr>
<tr>
<td>59</td>
<td>STALCTL</td>
</tr>
<tr>
<td>58</td>
<td>SYNCPR</td>
</tr>
<tr>
<td>57</td>
<td>TRCERR</td>
</tr>
<tr>
<td>56</td>
<td>RES0</td>
</tr>
<tr>
<td>55</td>
<td>EXLEVEL_NS_EL2</td>
</tr>
<tr>
<td>54</td>
<td>EXLEVEL_NS_EL1</td>
</tr>
<tr>
<td>53</td>
<td>EXLEVEL_NS_EL0</td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>NOOVERFLOW</td>
</tr>
<tr>
<td>28</td>
<td>NUMPROC[2:0]</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>RES0</td>
</tr>
<tr>
<td>25</td>
<td>RES0</td>
</tr>
<tr>
<td>24</td>
<td>RES0</td>
</tr>
<tr>
<td>23</td>
<td>RES0</td>
</tr>
<tr>
<td>22</td>
<td>RES0</td>
</tr>
<tr>
<td>21</td>
<td>RES0</td>
</tr>
<tr>
<td>20</td>
<td>RES0</td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

**NOOVERFLOW, bit [31]**

Indicates if overflow prevention is implemented.

<table>
<thead>
<tr>
<th>NOOVERFLOW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow prevention is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow prevention is implemented.</td>
</tr>
</tbody>
</table>

**NUMPROC[2:0], bits [30:28]**

This field is bits[2:0] of NUMPROC[4:0].

Indicates the number of PEs available for tracing.

<table>
<thead>
<tr>
<th>NUMPROC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>The trace unit can trace one PE.</td>
</tr>
</tbody>
</table>

This field reads as 0b00000.

The NUMPROC field is split as follows:

- NUMPROC[2:0] is TRCIDR3[30:28].
- NUMPROC[4:3] is TRCIDR3[13:12].
SYSSTALL, bit [27]

When TRCIDR3.STALLCTL == 0b1:

Indicates if stalling of the PE is permitted.

<table>
<thead>
<tr>
<th>SYSSTALL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stalling of the PE is not permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stalling of the PE is permitted.</td>
</tr>
</tbody>
</table>

The value of this field might be dynamic and change based on system conditions.

Otherwise:

Reserved, RES0.

STALLCTL, bit [26]

Indicates if trace unit implements stalling of the PE.

<table>
<thead>
<tr>
<th>STALLCTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stalling of the PE is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stalling of the PE is implemented.</td>
</tr>
</tbody>
</table>

SYNCPR, bit [25]

Indicates if an implementation has a fixed synchronization period.

<table>
<thead>
<tr>
<th>SYNCPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TRCSYNCPR is read-write so software can change the synchronization period.</td>
</tr>
<tr>
<td>0b1</td>
<td>TRCSYNCPR is read-only so the synchronization period is fixed.</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

TRCERR, bit [24]

Indicates forced tracing of System Error exceptions is implemented.

<table>
<thead>
<tr>
<th>TRCERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forced tracing of System Error exceptions is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forced tracing of System Error exceptions is implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

Indicates if Non-secure EL2 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL2 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL2 is implemented.</td>
</tr>
</tbody>
</table>

EXLEVEL_NS_EL1, bit [21]

Indicates if Non-secure EL1 implemented.
**EXLEVEL_NS_EL1**, bit [20]

Indicates if Non-secure EL1 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_NS_EL0, bit [19]**

Indicates if Non-secure EL0 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL0 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL0 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL3, bit [19]**

Indicates if Secure EL3 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL3 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL3 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL2, bit [18]**

Indicates if Secure EL2 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL2 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL2 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL1, bit [17]**

Indicates if Secure EL1 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL1 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL1 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL0, bit [16]**

Indicates if Secure EL0 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL0 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL0 is implemented.</td>
</tr>
</tbody>
</table>

**Bits [15:14]**

Reserved, RES0.

**NUMPROC[4:3], bits [13:12]**

This field is bits[4:3] of NUMPROC[4:0].

See NUMPROC[2:0] for the field description.

**CCITMIN, bits [11:0]**

When TRCIDR0.TRCCI == 0b1:

Indicates the minimum value that can be programmed in TRCCTLR.THRESHOLD.

When cycle counting in the instruction trace is supported, that is TRCIDR0.TRCCI == 0b1, then the minimum value of this field is 0x001.
Otherwise:

Reserved, RES0.

**Accessing the TRCIDR3**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1011</td>
<td>0b111</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR3;
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR3;
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR3;
    endif
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TRCIDR4 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR4 bits [31:0] are architecturally mapped to External register TRCIDR4[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR4 are UNDEFINED.

**Attributes**

TRCIDR4 is a 64-bit register.

**Field descriptions**

The TRCIDR4 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | NUMVMIDC | NUMCIDC | NUMSSCC | NUMRSPAIR | NUMPC | RES0 | SUPPDAC | NUMDVC | NUMACPAIRS |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**NUMVMIDC, bits [31:28]**

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

<table>
<thead>
<tr>
<th>NUMVMIDC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Virtual Context Identifier Comparators are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Virtual Context Identifier Comparator.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Virtual Context Identifier Comparators.</td>
</tr>
</tbody>
</table>

All other values are reserved.
**NUMCIDC, bits [27:24]**

Indicates the number of Context Identifier Comparators that are available for tracing.

<table>
<thead>
<tr>
<th>NUMCIDC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Context Identifier Comparators are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Context Identifier Comparator.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Context Identifier Comparators.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NUMSSCC, bits [23:20]**

Indicates the number of Single-shot Comparator Controls that are available for tracing.

<table>
<thead>
<tr>
<th>NUMSSCC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Single-shot Comparator Controls are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Single-shot Comparator Control.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Single-shot Comparator Controls.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NUMRSPAIR, bits [19:16]**

Indicates the number of resource selector pairs that are available for tracing.
### NUMRSPAIR

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>The implementation has zero resource selectors.</td>
<td>0b0000</td>
</tr>
<tr>
<td>The implementation has two resource selector pairs.</td>
<td>0b0001</td>
</tr>
<tr>
<td>The implementation has three resource selector pairs.</td>
<td>0b0010</td>
</tr>
<tr>
<td>The implementation has four resource selector pairs.</td>
<td>0b0011</td>
</tr>
<tr>
<td>The implementation has five resource selector pairs.</td>
<td>0b0100</td>
</tr>
<tr>
<td>The implementation has six resource selector pairs.</td>
<td>0b0101</td>
</tr>
<tr>
<td>The implementation has seven resource selector pairs.</td>
<td>0b0110</td>
</tr>
<tr>
<td>The implementation has eight resource selector pairs.</td>
<td>0b0111</td>
</tr>
<tr>
<td>The implementation has nine resource selector pairs.</td>
<td>0b1000</td>
</tr>
<tr>
<td>The implementation has ten resource selector pairs.</td>
<td>0b1001</td>
</tr>
<tr>
<td>The implementation has eleven resource selector pairs.</td>
<td>0b1010</td>
</tr>
<tr>
<td>The implementation has twelve resource selector pairs.</td>
<td>0b1011</td>
</tr>
<tr>
<td>The implementation has thirteen resource selector pairs.</td>
<td>0b1100</td>
</tr>
<tr>
<td>The implementation has fourteen resource selector pairs.</td>
<td>0b1101</td>
</tr>
<tr>
<td>The implementation has fifteen resource selector pairs.</td>
<td>0b1110</td>
</tr>
<tr>
<td>The implementation has sixteen resource selector pairs.</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

All other values are reserved.

### NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>No PE Comparator Inputs are available.</td>
<td>0b0000</td>
</tr>
<tr>
<td>The implementation has one PE Comparator Input.</td>
<td>0b0001</td>
</tr>
<tr>
<td>The implementation has two PE Comparator Inputs.</td>
<td>0b0010</td>
</tr>
<tr>
<td>The implementation has three PE Comparator Inputs.</td>
<td>0b0011</td>
</tr>
<tr>
<td>The implementation has four PE Comparator Inputs.</td>
<td>0b0100</td>
</tr>
<tr>
<td>The implementation has five PE Comparator Inputs.</td>
<td>0b0101</td>
</tr>
<tr>
<td>The implementation has six PE Comparator Inputs.</td>
<td>0b0110</td>
</tr>
<tr>
<td>The implementation has seven PE Comparator Inputs.</td>
<td>0b0111</td>
</tr>
<tr>
<td>The implementation has eight PE Comparator Inputs.</td>
<td>0b1000</td>
</tr>
</tbody>
</table>

All other values are reserved.

### Bits [11:9]

Reserved, RES0.

### SUPPDAC, bit [8]

When TRCIDR4.NUMACPAIRS ! = 0b0000:

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Binary Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data address comparisons not implemented.</td>
<td>0b0</td>
</tr>
<tr>
<td>Data address comparisons implemented.</td>
<td>0b1</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

Otherwise:

Reserved, RES0.

### NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.
<table>
<thead>
<tr>
<th>NUMDVC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No data value comparators implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>One data value comparator implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Two data value comparators implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Three data value comparators implemented.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Four data value comparators implemented.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Five data value comparators implemented.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Six data value comparators implemented.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Seven data value comparators implemented.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Eight data value comparators implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b0000.

**NUMACPAIRS, bits [3:0]**

Indicates the number of Address Comparator pairs that are available for tracing.

<table>
<thead>
<tr>
<th>NUMACPAIRS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Address Comparator pairs are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Address Comparator pair.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Address Comparator pairs.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Address Comparator pairs.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Address Comparator pairs.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Address Comparator pairs.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Address Comparator pairs.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Address Comparator pairs.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Address Comparator pairs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the TRCIDR4**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR4

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCIDR4;
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR4;
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR4;
    endif
TRCIDR5, ID Register 5

The TRCIDR5 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR5 bits [31:0] are architecturally mapped to External register TRCIDR5[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR5 are UNDEFINED.

**Attributes**

TRCIDR5 is a 64-bit register.

**Field descriptions**

The TRCIDR5 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Positions</th>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-31</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30-28</td>
<td>NUMCNTR</td>
<td>Indicates the number of Counters that are available for tracing.</td>
</tr>
<tr>
<td>27-25</td>
<td>NUMSEQSTATE</td>
<td>Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.</td>
</tr>
<tr>
<td>24-22</td>
<td>LPOVERRIDE</td>
<td>The Sequencer is not implemented.</td>
</tr>
<tr>
<td>21-18</td>
<td>ATBTRIG</td>
<td>Four Sequencer states are implemented.</td>
</tr>
<tr>
<td>17-14</td>
<td>TRACEIDSIZE</td>
<td>The Sequencer is not implemented.</td>
</tr>
<tr>
<td>13-10</td>
<td>NUMEXTINSEL</td>
<td>Four Sequencer states are implemented.</td>
</tr>
<tr>
<td>9-6</td>
<td>NUMEXTN</td>
<td>Four Sequencer states are implemented.</td>
</tr>
<tr>
<td>5-2</td>
<td></td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1-0</td>
<td></td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**NUMCNTR, bits [30:28]**

Indicates the number of Counters that are available for tracing.

<table>
<thead>
<tr>
<th>NUMCNTR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>No Counters are available.</td>
</tr>
<tr>
<td>0b001</td>
<td>One Counter implemented.</td>
</tr>
<tr>
<td>0b010</td>
<td>Two Counters implemented.</td>
</tr>
<tr>
<td>0b011</td>
<td>Three Counters implemented.</td>
</tr>
<tr>
<td>0b100</td>
<td>Four Counters implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.

**NUMSEQSTATE, bits [27:25]**

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

<table>
<thead>
<tr>
<th>NUMSEQSTATE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>The Sequencer is not implemented.</td>
</tr>
<tr>
<td>0b100</td>
<td>Four Sequencer states are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.
Bit [24]
Reserved, RES0.

LPOVERRIDE, bit [23]
Indicates support for Low-power Override Mode.

<table>
<thead>
<tr>
<th>LPOVERRIDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit does not support Low-power Override Mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit support Low-power Override Mode.</td>
</tr>
</tbody>
</table>

ATBTRIG, bit [22]
Indicates if the implementation can support ATB triggers.

<table>
<thead>
<tr>
<th>ATBTRIG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support ATB triggers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports ATB triggers.</td>
</tr>
</tbody>
</table>

This bit is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.

TRACEIDSIZE, bits [21:16]
Indicates the trace ID width.

<table>
<thead>
<tr>
<th>TRACEIDSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>The external trace interface is not implemented.</td>
</tr>
<tr>
<td>0b000111</td>
<td>The implementation supports a 7-bit trace ID.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Note that AMBA ATB requires a 7-bit trace ID width.

Bits [15:12]
Reserved, RES0.

NUMEXTINSEL, bits [11:9]
Indicates how many External Input Selector resources are implemented.

<table>
<thead>
<tr>
<th>NUMEXTINSEL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>No External Input Selector resources are available.</td>
</tr>
<tr>
<td>0b001</td>
<td>1 External Input Selector resource is available.</td>
</tr>
<tr>
<td>0b010</td>
<td>2 External Input Selector resources are available.</td>
</tr>
<tr>
<td>0b011</td>
<td>3 External Input Selector resources are available.</td>
</tr>
<tr>
<td>0b100</td>
<td>4 External Input Selector resources are available.</td>
</tr>
</tbody>
</table>

All other values are reserved.

NUMEXTIN, bits [8:0]
Indicates how many External Inputs are implemented.

<table>
<thead>
<tr>
<th>NUMEXTIN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11111111</td>
<td>Unified PMU event selection.</td>
</tr>
</tbody>
</table>

All other values are reserved.
## Accessing the TRCIDR5

Accesses to this register use the following encodings:

**MRS <Xt>, TRCIDR5**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCIDR5;
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR5;
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR5;
    endif
else
    return TRCIDR5;
endif
TRCIDR6, ID Register 6

The TRCIDR6 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR6 bits [31:0] are architecturally mapped to External register TRCIDR6[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR6 are **UNDEFINED**.

**Attributes**

TRCIDR6 is a 64-bit register.

**Field descriptions**

The TRCIDR6 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Reserved, RES0.

**Accessing the TRCIDR6**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR6

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1110</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR6;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR6;
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR6;
            end
    end

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TRCIDR7, ID Register 7

The TRCIDR7 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR7 bits [31:0] are architecturally mapped to External register TRCIDR7[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR7 are **UNDEFINED**.

**Attributes**

TRCIDR7 is a 64-bit register.

**Field descriptions**

The TRCIDR7 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 |

**Bits [63:0]**

Reserved, RES0.

**Accessing the TRCIDR7**

Accesses to this register use the following encodings:

\[\text{MRS} \ <Xt>, \ \text{TRCIDR7}\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1111</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR7;
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR7;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR7;
  endif
TRCIDR8, ID Register 8

The TRCIDR8 characteristics are:

**Purpose**

Returns the maximum speculation depth of the instruction trace element stream.

**Configuration**

AArch64 System register TRCIDR8 bits [31:0] are architecturally mapped to External register TRCIDR8[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR8 are UNDEFINED.

**Attributes**

TRCIDR8 is a 64-bit register.

**Field descriptions**

The TRCIDR8 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | MAXSPEC |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:32]**

Reserved, RES0.

**MAXSPEC, bits [31:0]**

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

**Accessing the TRCIDR8**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIDR8

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGRTR_EL2.TRCID == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR8;
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR8;
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCIDR8;

TRCIDR9, ID Register 9

The TRCIDR9 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

AArch64 System register TRCIDR9 bits [31:0] are architecturally mapped to External register TRCIDR9[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR9 are **UNDEFINED**.

**Attributes**

TRCIDR9 is a 64-bit register.

**Field descriptions**

The TRCIDR9 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>NUMP0KEY</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**NUMP0KEY, bits [31:0]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

**Otherwise:**

Reserved, RES0.

**Accessing the TRCIDR9**

Accesses to this register use the following encodings:

```
MRS <Xt>, TRCIDR9
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR9;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR9;
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR9;
            end if
        end if
    end if
end if

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TRCIMSPEC0, IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

**Purpose**

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

**Configuration**

AArch64 System register TRCIMSPEC0 bits [31:0] are architecturally mapped to External register `TRCIMSPEC0[31:0]`.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIMSPEC0 are UNDEFINED.

**Attributes**

TRCIMSPEC0 is a 64-bit register.

**Field descriptions**

The TRCIMSPEC0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>EN, bits [7:4]</td>
</tr>
<tr>
<td>30</td>
<td>When TRCIMSPEC0.SUPPORT != 0b0000:</td>
</tr>
<tr>
<td>29</td>
<td>Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.</td>
</tr>
<tr>
<td>28</td>
<td>Meaning</td>
</tr>
<tr>
<td>27</td>
<td>0b0000</td>
</tr>
<tr>
<td>26</td>
<td>0b0001</td>
</tr>
<tr>
<td>25</td>
<td>0b0010</td>
</tr>
<tr>
<td>24</td>
<td>0b0011</td>
</tr>
<tr>
<td>23</td>
<td>0b0100</td>
</tr>
<tr>
<td>22</td>
<td>0b0101</td>
</tr>
<tr>
<td>21</td>
<td>0b0110</td>
</tr>
<tr>
<td>20</td>
<td>0b0111</td>
</tr>
<tr>
<td>19</td>
<td>0b1000</td>
</tr>
<tr>
<td>18</td>
<td>0b1001</td>
</tr>
<tr>
<td>17</td>
<td>0b1010</td>
</tr>
<tr>
<td>16</td>
<td>0b1011</td>
</tr>
<tr>
<td>15</td>
<td>0b1100</td>
</tr>
<tr>
<td>14</td>
<td>0b1101</td>
</tr>
<tr>
<td>13</td>
<td>0b1110</td>
</tr>
<tr>
<td>12</td>
<td>0b1111</td>
</tr>
</tbody>
</table>
On a Trace unit reset, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**SUPPORT, bits [3:0]**

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

<table>
<thead>
<tr>
<th>SUPPORT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0011</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0100</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0101</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0110</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0111</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1000</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1001</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1010</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1011</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1100</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1101</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1110</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1111</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
</tbody>
</table>

Use of nonzero values requires written permission from Arm.

Access to this field is **RO**.

**Accessing the TRCIMSPEC0**

Accesses to this register use the following encodings:

MRS <Xt>, TRCIMSPEC0

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCIMSPECn == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    return TRCIMSPEC0;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elseif !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCIMSPEC0 = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCIMSPEC0 = X[t];
elseif PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRWTR_EL2.TRCIMSPECn == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TRCIMSPEC0 = X[t];
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCIMSPEC0 = X[t];
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCIMSPEC0 = X[t];

MSR TRCIMSPEC0, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRWTR_EL2.TRCIMSPECn == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TRCIMSPEC0 = X[t];
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCIMSPEC0 = X[t];
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCIMSPEC0 = X[t];
TRCIMSPEC<n>, IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

**Purpose**

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

**Configuration**

AArch64 System register TRCIMSPEC<n> bits [31:0] are architecturally mapped to External register TRCIMSPEC<n>[31:0].

This register is present only when the trace unit implements this OPTIONAL register and ETE is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are **UNDEFINED**.

**Attributes**

TRCIMSPEC<n> is a 64-bit register.

**Field descriptions**

The TRCIMSPEC<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**Accessing the TRCIMSPEC<n>**

Accesses to this register use the following encodings:

```
MRS <Xt>, TRCIMSPEC<n>
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0:n[2:0]</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIMSPEC[UInt(CRm<2:0>)];
    elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIMSPEC[UInt(CRm<2:0>)];
    elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIMSPEC[UInt(CRm<2:0>)];

MSR TRCIMSPEC<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0:n&lt;2:0&gt;</td>
<td>0b11</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCIMSPECn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCIMSPEC[UInt(CRm<2:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCIMSPEC[UInt(CRm<2:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCIMSPEC[UInt(CRm<2:0>)] = X[t];
The TRCOSLSR characteristics are:

**Purpose**

Returns the status of the Trace OS Lock.

**Configuration**

AArch64 System register TRCOSLSR bits [31:0] are architecturally mapped to External register \( \text{TRCOSLSR}[31:0] \).

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCOSLSR are \textbf{UNDEFINED}.

**Attributes**

TRCOSLSR is a 64-bit register.

**Field descriptions**

The TRCOSLSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>OSLM[0]</td>
<td>Trace OS Lock model.</td>
</tr>
</tbody>
</table>

The OSLM field is split as follows:

- OSLM[2:1] is TRCOSLSR[4:3].
- OSLM[0] is TRCOSLSR[0].

**Bit [2]**

Reserved, RES0.
OSLK, bit [1]

OS Lock status.

<table>
<thead>
<tr>
<th>OSLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The OS Lock is unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>The OS Lock is locked.</td>
</tr>
</tbody>
</table>

Note that this field indicates the state of the PE OS Lock.

OSLM[0], bit [0]

This field is bit[0] of OSLM[2:0].

See OSLM[2:1] for the field description.

Accessing the TRCOSLSR

Accesses to this register use the following encodings:

MRS <Xt>, TRCOSLSR

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b10</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if CPACR_EL1.TTA == '1' then
      AArch64.SystemAccessTrap(EL1, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
       HDFGRTR_EL2.TRCSLSR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      return TRCOSLSR;
   endif
elsif PSTATE.EL == EL2 then
   if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   else
      return TRCOSLSR;
   endif
elsif PSTATE.EL == EL3 then
   if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   else
      return TRCOSLSR;
   endif
else
   return TRCOSLSR;
endif
TRCPRGCTRLR, Programming Control Register

The TRCPRGCTRLR characteristics are:

**Purpose**

Enables the trace unit.

**Configuration**

AArch64 System register TRCPRGCTRLR bits [31:0] are architecturally mapped to External register TRCPRGCTRLR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPRGCTRLR are undefined.

**Attributes**

TRCPRGCTRLR is a 64-bit register.

**Field descriptions**

The TRCPRGCTRLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>EN, Trace unit enable.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to 0.

**Accessing the TRCPRGCTRLR**

Must be programmed.

Accesses to this register use the following encodings:

**MRS <Xt>, TRCPRGCTRLR**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if CPACR_EL1.TTA == '1' then
      AArch64.SystemAccessTrap(EL1, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCPRGCTLR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   else
      return TRCPRGCTLR;
   elsif PSTATE.EL == EL2 then
      if CPTR_EL2.TTA == '1' then
         AArch64.SystemAccessTrap(EL2, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         return TRCPRGCTLR;
      elsif PSTATE.EL == EL3 then
         if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
         else
            return TRCPRGCTLR;
      end
end

MSR TRCPRGCTLR, <xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if CPACR_EL1.TTA == '1' then
      AArch64.SystemAccessTrap(EL1, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCPRGCTLR == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   else
      TRCPRGCTLR = X[t];
   elsif PSTATE.EL == EL2 then
      if CPTR_EL2.TTA == '1' then
         AArch64.SystemAccessTrap(EL2, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         TRCPRGCTLR = X[t];
      elsif PSTATE.EL == EL3 then
         if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
         else
            TRCPRGCTLR = X[t];
      end
end
TRCQCTRL, Q Element Control Register

The TRCQCTRL characteristics are:

**Purpose**

Controls when Q elements are enabled.

**Configuration**

AArch64 System register TRCQCTRL bits [31:0] are architecturally mapped to External register TRCQCTRL[31:0].

This register is present only when ETE is implemented and TRCIDR0.QFILT == 0b1. Otherwise, direct accesses to TRCQCTRL are **UNDEFINED**.

**Attributes**

TRCQCTRL is a 64-bit register.

**Field descriptions**

The TRCQCTRL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>MODE</td>
<td>Selects whether the Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:</td>
</tr>
<tr>
<td>30</td>
<td>RANGE</td>
<td>Specifies the Address Range Comparators to be used for controlling Q elements.</td>
</tr>
</tbody>
</table>

**Bits [63:9]**

Reserved, RES0.

**MODE, bit [8]**

Selects whether the Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

<table>
<thead>
<tr>
<th>MODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exclude mode. The Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map.</td>
</tr>
<tr>
<td>0b1</td>
<td>Include Mode. The Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0b0 then Q elements are disabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**RANGE<m>, bit [m], for m = 0 to 7**

Specifies the Address Range Comparators to be used for controlling Q elements.
### RANGE<\(m\)>

<table>
<thead>
<tr>
<th>(\text{Meaning} )</th>
<th>(\text{RANGE&lt;}m\text{&gt;} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The address range that Address Range Comparator (m) defines, is not selected.</td>
<td>0b0</td>
</tr>
<tr>
<td>The address range that Address Range Comparator (m) defines, is selected.</td>
<td>0b1</td>
</tr>
</tbody>
</table>

This bit is \(\text{RES}0\) if \(m \geq \text{TRCIDR4.NUMACPAIRS}\).

On a Trace unit reset, this field resets to an architecturally \textit{UNKNOWN} value.

### Accessing the TRCQCTRLR

Must be programmed if \text{TRCCONFIGR.}QE \(!= 0b00\).

Writes are \textit{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

**MRS \(<Xt>, \text{TRCQCTRLR}\)**

<table>
<thead>
<tr>
<th>(\text{op0} )</th>
<th>(\text{op1} )</th>
<th>(\text{CRn} )</th>
<th>(\text{CRm} )</th>
<th>(\text{op2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if \(\text{PSTATE.EL} == \text{EL0}\) then
UNDEFINED;
elseif \(\text{PSTATE.EL} == \text{EL1}\) then
  if \(\text{CPACR_EL1.TTA} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL1, 0x10)};
  elseif \(\text{EL2Enabled()} \&\& \!\text{ELUsingAArch32(EL2)} \&\& \text{CPTR_EL2.TTA} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL2, 0x10)};
  elseif \(\text{EL2Enabled()} \&\& \!\text{ELUsingAArch32(EL2)} \&\& (\!\text{HaveEL(EL3)} || \text{SCR_EL3.FGTEn} == '1') \&\& \text{HDFGTR_EL2.TRC} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL2, 0x10)};
  elseif \(\text{HaveEL(EL3)} \&\& \!\text{ELUsingAArch32(EL3)} \&\& \text{CPTR_EL3.TTA} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL3, 0x10)};
  else
    return TRCQCTRLR;
elseif \(\text{PSTATE.EL} == \text{EL2}\) then
  if \(\text{CPTR_EL2.TTA} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL2, 0x10)};
  elseif \(\text{HaveEL(EL3)} \&\& \!\text{ELUsingAArch32(EL3)} \&\& \text{CPTR_EL3.TTA} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL3, 0x10)};
  else
    return TRCQCTRLR;
elseif \(\text{PSTATE.EL} == \text{EL3}\) then
  if \(\text{CPTR_EL3.TTA} == '1'\) then
    \text{AArch64.SystemAccessTrap(EL3, 0x10)};
  else
    return TRCQCTRLR;
else
  return TRCQCTRLR;
```

**MSR \(\text{TRCQCTRLR, <Xt>}\)**

<table>
<thead>
<tr>
<th>(\text{op0} )</th>
<th>(\text{op1} )</th>
<th>(\text{CRn} )</th>
<th>(\text{CRm} )</th>
<th>(\text{op2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCQCTRL = X[t];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCQCTRL = X[t];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCQCTRL = X[t];
    end
TRCRSCTLR<n>, Resource Selection Control Register <n>, n = 2 - 31

The TRCRSCTLR<n> characteristics are:

**Purpose**

Controls the selection of the resources in the trace unit.

**Configuration**

AArch64 System register TRCRSCTLR<n> bits [31:0] are architecturally mapped to External register TRCRSCTLR<n>[31:0].

This register is present only when ETE is implemented and ((AArch64-TRCIDR4.NUMRSPAIR + 1) * 2) > n. Otherwise, direct accesses to TRCRSCTLR<n> are UNDEFINED.

Resource selector 0 always returns FALSE.

Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

**Attributes**

TRCRSCTLR<n> is a 64-bit register.

**Field descriptions**

The TRCRSCTLR<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Bits [63:22]**

Reserved, RES0.

**PAIRINV, bit [21]**

For TRCRSCTLR<n>, where n is even, controls whether the combined result from a resource selector pair is inverted.

<table>
<thead>
<tr>
<th>PAIRINV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not invert the combined output of the 2 resource selectors.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invert the combined output of the 2 resource selectors.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**INV, bit [20]**

Controls whether the resource, that GROUP and SELECT selects, is inverted.
<table>
<thead>
<tr>
<th>INV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not invert the output of this selector.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invert the output of this selector.</td>
</tr>
</tbody>
</table>

If:

- A is the register TRCRSCTLR< m > where m is even.
- B is the register TRCRSCTLR< m+1 >.

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- 0b000 -> A and B.
- 0b001 -> RESERVED.
- 0b010 -> not(A) and B.
- 0b011 -> not(A) and not(B).
- 0b100 -> not(A) or not(B).
- 0b101 -> not(A) or B.
- 0b110 -> RESERVED.
- 0b111 -> A or B.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**GROUP, bits [19:16]**

Selects a group of resources.

<table>
<thead>
<tr>
<th>GROUP</th>
<th>Meaning</th>
<th>SELECT encoding for External Input Selectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>External Input Selectors.</td>
<td></td>
</tr>
<tr>
<td>0b0001</td>
<td>PE Comparator Inputs.</td>
<td></td>
</tr>
<tr>
<td>0b0010</td>
<td>Counters and Sequencer.</td>
<td></td>
</tr>
<tr>
<td>0b0011</td>
<td>Single-shot Comparator Controls.</td>
<td></td>
</tr>
<tr>
<td>0b0100</td>
<td>Single Address Comparators.</td>
<td></td>
</tr>
<tr>
<td>0b0101</td>
<td>Address Range Comparators.</td>
<td></td>
</tr>
<tr>
<td>0b0110</td>
<td>Context Identifier Comparators.</td>
<td></td>
</tr>
<tr>
<td>0b0111</td>
<td>Virtual Context Identifier Comparators.</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**SELECT, bits [15:0]**

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.
EXTIN\(<m>\), bit \([m]\), for \(m = 0\) to \(3\)

Selects one or more External Inputs.

<table>
<thead>
<tr>
<th>EXTIN(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore EXTIN (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select EXTIN (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**SELECT encoding for PE Comparator Inputs**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>PECOMP(&lt;m&gt;), bit ([m])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [15:8]

Reserved, RES0.

PECOMP\(<m>\), bit \([m]\), for \(m = 0\) to \(7\)

Selects one or more PE Comparator Inputs.

<table>
<thead>
<tr>
<th>PECOMP(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore PE Comparator Input (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select PE Comparator Input (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**SELECT encoding for Counters and Sequencer**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>SEQUENCER(&lt;m&gt;), bit ([m+4])</td>
<td>COUNTERS(&lt;m&gt;), bit ([m])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [15:8]

Reserved, RES0.

SEQUENCER\(<m>\), bit \([m+4]\), for \(m = 0\) to \(3\)

Sequencer states.

<table>
<thead>
<tr>
<th>SEQUENCER(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Sequencer state (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Sequencer state (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS\(<m>\), bit \([m]\), for \(m = 0\) to \(3\)

Counters resources at zero.

<table>
<thead>
<tr>
<th>COUNTERS(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Counter (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Counter (m) is zero.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**SELECT encoding for Single-shot Comparator Controls**


### Bits [15:8]

Reserved, **RES0**.

### **SINGLE_SHOT<m>, bit [m]**, for m = 0 to 7

Selects one or more Single-shot Comparator Controls.

<table>
<thead>
<tr>
<th><strong>SINGLE_SHOT&lt;m&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>Ignore Single-shot Comparator Control m.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>Select Single-shot Comparator Control m.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### SELECT encoding for Single Address Comparators

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SAC&lt;m&gt;, bit [m]</strong></td>
</tr>
</tbody>
</table>

### **SAC<m>, bit [m]**, for m = 0 to 15

Selects one or more Single Address Comparators.

<table>
<thead>
<tr>
<th><strong>SAC&lt;m&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>Ignore Single Address Comparator m.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>Select Single Address Comparator m.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### SELECT encoding for Address Range Comparators

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ARC&lt;m&gt;, bit [m]</strong></td>
</tr>
</tbody>
</table>

### Bits [15:8]

Reserved, **RES0**.

### **ARC<m>, bit [m]**, for m = 0 to 7

Selects one or more Address Range Comparators.

<table>
<thead>
<tr>
<th><strong>ARC&lt;m&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>Ignore Address Range Comparator m.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>Select Address Range Comparator m.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### SELECT encoding for Context Identifier Comparators

<table>
<thead>
<tr>
<th>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CID&lt;m&gt;, bit [m]</strong></td>
</tr>
</tbody>
</table>

### Bits [15:8]

Reserved, **RES0**.
**CID<sub>m>, bit [m], for m = 0 to 7**

Selects one or more Context Identifier Comparators.

<table>
<thead>
<tr>
<th>CID&lt;sub&gt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Context Identifier Comparator m.</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Context Identifier Comparator m.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally unknown value.

**SELECT encoding for Virtual Context Identifier Comparators**

<table>
<thead>
<tr>
<th></th>
<th>VMID&lt;sub&gt;m&gt;, bit [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**Bits [15:8]**

Reserved, RES0.

**VMID<sub>m>, bit [m], for m = 0 to 7**

Selects one or more Virtual Context Identifier Comparators.

<table>
<thead>
<tr>
<th>VMID&lt;sub&gt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Virtual Context Identifier Comparator m.</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Virtual Context Identifier Comparator m.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally unknown value.

**Accessing the TRCRSCTLR<n>**

Must be programmed if any of the following are true:

- TRCCNTCTRL<sub>a>.RLDEVENT.TYPE == 0b0 and TRCCNTCTRL<sub>a>.RLDEVENT.SEL == n.
- TRCCNTCTRL<sub>a>.RLDEVENT.TYPE == 0b1 and TRCCNTCTRL<sub>a>.RLDEVENT.SEL == n/2.
- TRCCNTCTRL<sub>a>.CNTEVENT.TYPE == 0b0 and TRCCNTCTRL<sub>a>.CNTEVENT.SEL == n.
- TRCCNTCTRL<sub>a>.CNTEVENT.TYPE == 0b1 and TRCCNTCTRL<sub>a>.CNTEVENT.SEL == n/2.
- TRCEVENTCTLR0.EVENT0.TYPE == 0b0 and TRCEVENTCTLR0.EVENT0.SEL == n.
- TRCEVENTCTLR0.EVENT0.TYPE == 0b1 and TRCEVENTCTLR0.EVENT0.SEL == n/2.
- TRCEVENTCTLR0.EVENT1.TYPE == 0b0 and TRCEVENTCTLR0.EVENT1.SEL == n.
- TRCEVENTCTLR0.EVENT1.TYPE == 0b1 and TRCEVENTCTLR0.EVENT1.SEL == n/2.
- TRCEVENTCTLR0.EVENT2.TYPE == 0b0 and TRCEVENTCTLR0.EVENT2.SEL == n.
- TRCEVENTCTLR0.EVENT2.TYPE == 0b1 and TRCEVENTCTLR0.EVENT2.SEL == n/2.
- TRCEVENTCTLR0.EVENT3.TYPE == 0b0 and TRCEVENTCTLR0.EVENT3.SEL == n.
- TRCEVENTCTLR0.EVENT3.TYPE == 0b1 and TRCEVENTCTLR0.EVENT3.SEL == n/2.
- TRCSEQEVR<sub>a>.B.TYPE == 0b0 and TRCSEQEVR<sub>a>.B.SEL == n.
- TRCSEQEVR<sub>a>.B.TYPE == 0b1 and TRCSEQEVR<sub>a>.B.SEL == n/2.
- TRCSEQEVR<sub>a>.F.TYPE == 0b0 and TRCSEQEVR<sub>a>.F.SEL == n.
- TRCSEQEVR<sub>a>.F.TYPE == 0b1 and TRCSEQEVR<sub>a>.F.SEL == n/2.
- TRCSEQRST.<sub>TEVR.RST.TYPE == 0b0 and TRCSEQRST.<sub>TEVR.RST.SEL == n.
• TRCSEQRSTEVR.RST.TYPE == 0b1 and TRCSEQRSTEVR.RST.SEL == n/2.
• TRCTSCTRL.EVENT.TYPE == 0b0 and TRCTSCTRL.EVENT.SEL == n.
• TRCTSCTRL.EVENT.TYPE == 0b1 and TRCTSCTRL.EVENT.SEL == n/2.
• TRCVICTLR.EVENT.TYPE == 0b0 and TRCVICTLR.EVENT.SEL == n.
• TRCVICTLR.EVENT.TYPE == 0b1 and TRCVICTLR.EVENT.SEL == n/2.

Writes are constrained unpredictable if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCRSCTRL<n>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b0001</td>
<td>0b0001</td>
<td>n[3:0]</td>
<td>0b00:n[4]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCRSCTRL[UInt(op2<0>:CRm<3:0>)];
    end if;
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCRSCTRL[UInt(op2<0>:CRm<3:0>)];
    end if;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCRSCTRL[UInt(op2<0>:CRm<3:0>)];
    end if;
else
    MSR TRCRSCTRL<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b0001</td>
<td>0b0001</td>
<td>n[3:0]</td>
<td>0b00:n[4]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.SystemAccessTrap(EL2, 0x18);
    endif
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCRSCTLR[UInt(op2<0>:CRm<3:0>)] = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCRSCTLR[UInt(op2<0>:CRm<3:0>)] = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCRSCTLR[UInt(op2<0>:CRm<3:0>)] = X[t];
    endif
TRCRSR, Resources Status Register

The TRCRSR characteristics are:

**Purpose**

Use this to set, or read, the status of the resources.

**Configuration**

AArch64 System register TRCRSR bits [31:0] are architecturally mapped to External register TRCRSR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCRSR are UNDEFINED.

**Attributes**

TRCRSR is a 64-bit register.

**Field descriptions**

The TRCRSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>TA, bit [12]</td>
</tr>
<tr>
<td>11-0</td>
<td>EVENT(&lt;m&gt;), bit [m+8], for m = 0 to 3</td>
</tr>
</tbody>
</table>

**Bits [63:13]**

Reserved, RES0.

**TA, bit [12]**

Tracing active.

<table>
<thead>
<tr>
<th>TA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing is not active.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing is active.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**EVENT\(<m>\), bit [m+8], for m = 0 to 3**

Untraced status of ETEEEvents.

<table>
<thead>
<tr>
<th>EVENT(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An ETEEEvent[n] has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>An ETEEEvent[n] has occurred while the resources were in the Paused state.</td>
</tr>
</tbody>
</table>

This bit is RES0 if TRCIDR4.NUMRSPAIR == 0 \(\|\) m > TRCIDR0.NUMEVENT.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
**Bits [7:4]**

Reserved, RES0.

**EXTIN<m>, bit [m], for m = 0 to 3**

The sticky status of the External Input Selectors.

<table>
<thead>
<tr>
<th>EXTIN&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>An event selected by External Input Selector[n] has not occurred.</td>
</tr>
<tr>
<td>01</td>
<td>At least one event selected by External Input Selector[n] has occurred while the resources were in the Paused state.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR5.NUMEXTINSEL.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCRSR**

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings:

**MRS <Xt>, TRCRSR**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b001</td>
<td>0b0000</td>
<td>0b1010</td>
<td>0b000</td>
<td></td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if CPACR_EL1.TTA == '1' then
      AArch64.SystemAccessTrap(EL1, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
   else
      return TRCRSR;
   elsif PSTATE.EL == EL2 then
      if CPTR_EL2.TTA == '1' then
         AArch64.SystemAccessTrap(EL2, 0x18);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         return TRCRSR;
   elsif PSTATE.EL == EL3 then
      if CPTR_EL3.TTA == '1' then
         AArch64.SystemAccessTrap(EL3, 0x18);
      else
         return TRCRSR;
if PSTATE_EL == EL0 then
  UNDEFINED;
elsif PSTATE_EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HDFGWTR_EL2.TRCE == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCRSR = X[t];
  elsif PSTATE_EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCRSR = X[t];
  elsif PSTATE_EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCRSR = X[t];
The TRCSEQEVR\textsubscript{n} characteristics are:

**Purpose**

Moves the Sequencer state:

- Backwards, from state \(n+1\) to state \(n\) when a programmed resource event occurs.
- Forwards, from state \(n\) to state \(n+1\) when a programmed resource event occurs.

**Configuration**

AArch64 System register TRCSEQEVR\textsubscript{n} bits [31:0] are architecturally mapped to External register TRCSEQEVR\textsubscript{n}[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQEVR\textsubscript{n} are **UNDEFINED**.

**Attributes**

TRCSEQEVR\textsubscript{n} is a 64-bit register.

**Field descriptions**

The TRCSEQEVR\textsubscript{n} bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:16]</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

Backward field. Defines whether the backward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state \(n+1\) to state \(n\). For example, if TRCSEQEVR2.B.SEL == 0x14 then when event 0x14 occurs, the Sequencer moves from state 3 to state 2.

<table>
<thead>
<tr>
<th>B_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCSEQEVR\textsubscript{n}.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCSEQEVR\textsubscript{n}.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR\textsubscript{n}.B.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.
Bits [14:13]

Reserved, RES0.

B_SEL, bits [12:8]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.B.TYPE controls whether TRCSEQEVR<n>.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Backward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

F_TYPE, bit [7]

Chooses the type of Resource Selector.

Backward field. Defines whether the forward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == 0x12 then when event 0x12 occurs, the Sequencer moves from state 1 to state 2.

<table>
<thead>
<tr>
<th>F_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCSEQEVR&lt;n&gt;.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCSEQEVR&lt;n&gt;.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR&lt;n&gt;.F.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

F_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.F.TYPE controls whether TRCSEQEVR<n>.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Forward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCSEQEVR<n>

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCSEQEVR<n>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>

Page 1889
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQEVR[UInt(CRm<1:0>)];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQEVR[UInt(CRm<1:0>)];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQEVR[UInt(CRm<1:0>)];
    end
MSR TRCSEQEVR<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b00:n[1:0]</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQEVR[UInt(CRm<1:0>)] = X[t];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQEVR[UInt(CRm<1:0>)] = X[t];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQEVR[UInt(CRm<1:0>)] = X[t];
    end

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe34331cc1e9707
TRCSEQRSTEVR, Sequencer Reset Control Register

The TRCSEQRSTEVR characteristics are:

**Purpose**

Moves the Sequencer to state 0 when a programmed resource event occurs.

**Configuration**

AArch64 System register TRCSEQRSTEVR bits [31:0] are architecturally mapped to External register TRCSEQRSTEVR[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQRSTEVR are UNDEFINED.

**Attributes**

TRCSEQRSTEVR is a 64-bit register.

**Field descriptions**

The TRCSEQRSTEVR bit assignments are:

<table>
<thead>
<tr>
<th>Bit(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>RST_TYPE, bit [7]</td>
</tr>
<tr>
<td>6:5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4:0</td>
<td>RST_SEL, bits [4:0]</td>
</tr>
</tbody>
</table>

**RST_TYPE, bit [7]**

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>RST_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCSEQRSTEVR.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCSEQRSTEVR.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEVR.RST.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [6:5]**

Reserved, RES0.

**RST_SEL, bits [4:0]**

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEVR.RST.TYPE controls whether TRCSEQRSTEVR.RST.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.
If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is **UNPREDICTABLE**, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the TRCSEQRSTEVR**

Must be programmed if TRCRSCTL<ae>.GROUP == 0b0010 and TRCRSCTL<ae>.SEQUENCER != 0b0000.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

**MRS <Xt>, TRCSEQRSTEVR**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCSEQRSTEVR;
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQRSTEVR;
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQRSTEVR;
    endif
endif
```

**MSR TRCSEQRSTEVR, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQRSTEVR = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQRSTEVR = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQRSTEVR = X[t];
    endif
else
    TRCSEQRSTEVR = X[t];
endif
TRCSEQSTR, Sequencer State Register

The TRCSEQSTR characteristics are:

**Purpose**

Use this to set, or read, the Sequencer state.

**Configuration**

AArch64 System register TRCSEQSTR bits [31:0] are architecturally mapped to External register TRCSEQSTR[31:0].

This register is present only when ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQSTR are **UNDEFINED**.

**Attributes**

TRCSEQSTR is a 64-bit register.

**Field descriptions**

The TRCSEQSTR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>62</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>61</td>
<td>STATE</td>
</tr>
<tr>
<td>60</td>
<td>Reserved</td>
</tr>
<tr>
<td>59</td>
<td>Reserved</td>
</tr>
<tr>
<td>58</td>
<td>Reserved</td>
</tr>
<tr>
<td>57</td>
<td>Reserved</td>
</tr>
<tr>
<td>56</td>
<td>Reserved</td>
</tr>
<tr>
<td>55</td>
<td>Reserved</td>
</tr>
<tr>
<td>54</td>
<td>Reserved</td>
</tr>
<tr>
<td>53</td>
<td>Reserved</td>
</tr>
<tr>
<td>52</td>
<td>Reserved</td>
</tr>
<tr>
<td>51</td>
<td>Reserved</td>
</tr>
<tr>
<td>50</td>
<td>Reserved</td>
</tr>
<tr>
<td>49</td>
<td>Reserved</td>
</tr>
<tr>
<td>48</td>
<td>Reserved</td>
</tr>
<tr>
<td>47</td>
<td>Reserved</td>
</tr>
<tr>
<td>46</td>
<td>Reserved</td>
</tr>
<tr>
<td>45</td>
<td>Reserved</td>
</tr>
<tr>
<td>44</td>
<td>Reserved</td>
</tr>
<tr>
<td>43</td>
<td>Reserved</td>
</tr>
<tr>
<td>42</td>
<td>Reserved</td>
</tr>
<tr>
<td>41</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Reserved</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>38</td>
<td>Reserved</td>
</tr>
<tr>
<td>37</td>
<td>Reserved</td>
</tr>
<tr>
<td>36</td>
<td>Reserved</td>
</tr>
<tr>
<td>35</td>
<td>Reserved</td>
</tr>
<tr>
<td>34</td>
<td>Reserved</td>
</tr>
<tr>
<td>33</td>
<td>Reserved</td>
</tr>
<tr>
<td>32</td>
<td>Reserved</td>
</tr>
<tr>
<td>31</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [63:2]**

Reserved, RES0.

**STATE, bits [1:0]**

Set or returns the state of the Sequencer.

<table>
<thead>
<tr>
<th>STATE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>State 0.</td>
</tr>
<tr>
<td>0b01</td>
<td>State 1.</td>
</tr>
<tr>
<td>0b10</td>
<td>State 2.</td>
</tr>
<tr>
<td>0b11</td>
<td>State 3.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the TRCSEQSTR**

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

Reads from this register might return an **UNKNOWN** value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x10);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.TRCSEQSTR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQSTR;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQSTR = X[t];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQSTR = X[t];
elseif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x10);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HDFGRTR_EL2.TRCSEQSTR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQSTR = X[t];
elseif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQSTR = X[t];
elseif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQSTR = X[t];
TRCSSCCR\(<n>\), Single-shot Comparator Control Register \(<n>\), n = 0 - 7

The TRCSSCCR\(<n>\) characteristics are:

**Purpose**

Controls the corresponding Single-shot Comparator Control resource.

**Configuration**

AArch64 System register TRCSSCCR\(<n>\) bits [31:0] are architecturally mapped to External register TRCSSCCR\(<n>[31:0]\).

This register is present only when ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCCR\(<n>\) are undefined.

**Attributes**

TRCSSCCR\(<n>\) is a 64-bit register.

**Field descriptions**

The TRCSSCCR\(<n>\) bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
<td>Reserved, Reserved.</td>
</tr>
<tr>
<td>62</td>
<td>RST</td>
<td>Selects the Single-shot Comparator Control mode.</td>
</tr>
<tr>
<td>61</td>
<td>ARC&lt;0&gt;, bit [16]</td>
<td>Selects one or more Address Range Comparators.</td>
</tr>
<tr>
<td>59</td>
<td>ARC&lt;1&gt;, bit [16]</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>ARC&lt;2&gt;, bit [16]</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>ARC&lt;3&gt;, bit [16]</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>ARC&lt;4&gt;, bit [16]</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>ARC&lt;5&gt;, bit [16]</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>ARC&lt;6&gt;, bit [16]</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>ARC&lt;7&gt;, bit [16]</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:25]**

Reserved, RES0.

**RST, bit [24]**

Selects the Single-shot Comparator Control mode.

<table>
<thead>
<tr>
<th>RST</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single-shot Comparator Control is in single-shot mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single-shot Comparator Control is in multi-shot mode.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**ARC<\(m\)>, bit [\(m+16\)]**, for \(m = 0\) to 7

Selects one or more Address Range Comparators for Single-shot control.

<table>
<thead>
<tr>
<th>ARC&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Range Comparator (m), is not selected for Single-shot control.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Range Comparator (m), is selected for Single-shot control.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \(m \geq\) TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
**SAC\(<m>\), bit [m], for m = 0 to 15**

Selects one or more Single Address Comparators for Single-shot control.

<table>
<thead>
<tr>
<th>SAC(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single Address Comparator (m), is not selected for Single-shot control.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single Address Comparator (m), is selected for Single-shot control.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \(m \geq 2 \times \text{TRCIDI4[NUMACPAIRS]}\).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the TRCSSCCR\(<n>\)**

Must be programmed if any TRCRSCTLR\(<a>\).GROUP == 0b0011 and TRCRSCTLR\(<a>\).SINGLE_SHOT[\(n\)] == 0b1.

Writes are \texttt{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

**MRS \(<Xt>, TRCSSCCR\(<n>\)**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b0:n[2:0]</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then UNDEFINED;

elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSSCCR[UInt(CRm<2:0>)];
  endif

elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSSCCR[UInt(CRm<2:0>)];
  endif

elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSSCCR[UInt(CRm<2:0>)];
  endif

MSR TRCSSCCR\(<n>, \(<Xt>\)**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b0:n[2:0]</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TRCSSCCR[UInt(CRm<2:0>)] = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSCCR[UInt(CRm<2:0>)] = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSCCR[UInt(CRm<2:0>)] = X[t];
    endif
TRCSSCSR\(<n>\), Single-shot Comparator Control Status Register \(<n>\), n = 0 - 7

The TRCSSCSR\(<n>\) characteristics are:

**Purpose**

Returns the status of the corresponding Single-shot Comparator Control.

**Configuration**

AArch64 System register TRCSSCSR\(<n>\) bits [31:0] are architecturally mapped to External register TRCSSCSR\(<n>[31:0]\).

This register is present only when ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCSR\(<n>\) are UNDEFINED.

**Attributes**

TRCSSCSR\(<n>\) is a 64-bit register.

**Field descriptions**

The TRCSSCSR\(<n>\) bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Status, bit [31].</td>
</tr>
<tr>
<td>61 - 32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>Pending, bit [30].</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29 - 0</td>
<td>Reserved, PCDVDAIENST.</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**STATUS, bit [31]**

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by TRCSSCCR\(<n>\).ARC, TRCSSCCR\(<n>\).SAC, and TRCSSPCICR\(<n>\).PC.

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No match has occurred. When the first match occurs, this field takes a value of 0b1. It remains at 0b1 until explicitly modified by a write to this register.</td>
</tr>
</tbody>
</table>
| 0b1    | One or more matches has occurred. If TRCSSCCR\(<n>\).RST == 0b0 then:  
• There is only one match and no more matches are possible.  
• Software must reset this bit to 0b0 to re-enable the Single-shot Comparator Control. |

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**PENDING, bit [30]**

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.
### PENDING

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### Bits [29:4]

Reserved, RES0.

### PC, bit [3]

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

Access to this field is **RO**.

### DV, bit [2]

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

Access to this field is **RO**.

### DA, bit [1]

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

Access to this field is **RO**.

### INST, bit [0]

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.
This Single-shot Comparator Control does not support instruction address comparisons.

This Single-shot Comparator Control supports instruction address comparisons. This bit reads as 0b1.

Access to this field is RO.

### Accessing the TRCSSCSR<n>

Must be programmed if TRCRSCTL<e>.GROUP == 0b0011 and TRCRSCTL<e>.SINGLE_SHOT[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings:

**MRS <Xt>, TRCSSCSR<n>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b1:n[2:0]</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCSSCSRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSSCSR[UInt(CRm<2:0>)];
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSCSR[UInt(CRm<2:0>)];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSCSR[UInt(CRm<2:0>)];

**MSR TRCSSCSR<n>, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b1:n[2:0]</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCSSCSRn == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TRCSSCSR[UInt(CRm<2:0>)] = X[t];
    endif
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSCSR[UInt(CRm<2:0>)] = X[t];
    endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSCSR[UInt(CRm<2:0>)] = X[t];
    endif
The TRCSSPCICR\(n\) characteristics are:

**Purpose**

Returns the status of the corresponding Single-shot Comparator Control.

**Configuration**

AArch64 System register TRCSSPCICR\(n\) bits [31:0] are architecturally mapped to External register TRCSSPCICR\(n\)[31:0].

This register is present only when ETE is implemented, TRCIDR4.NUMSSCC > \(n\), TRCIDR4.NUMPC > 0b0000 and TRCSSCSR\(n\).PC == 0b1. Otherwise, direct accesses to TRCSSPCICR\(n\) are UNDEFINED.

**Attributes**

TRCSSPCICR\(n\) is a 64-bit register.

**Field descriptions**

The TRCSSPCICR\(n\) bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | PC\(m\), bit [m] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:8]**

Reserved, RES0.

**PC\(m\), bit [m], for m = 0 to 7**

Selects one or more PE Comparator Inputs for Single-shot control.

<table>
<thead>
<tr>
<th>PC(m)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The single PE Comparator Input m, is not selected as for Single-shot control.</td>
</tr>
<tr>
<td>0b1</td>
<td>The single PE Comparator Input m, is selected as for Single-shot control.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \(m \geq \) TRCIDR4.NUMPC.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCSSPCICR\(n\)**

Must be programmed if implemented and any TRCRSCTLR\(a\).GROUP == 0b0011 and TRCRSCTLR\(a\).SINGLE_SHOT[\(n\)] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.
Accesses to this register use the following encodings:

**MRS <Xt>, TRCSSPCICR<n>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b0:n[2:0]</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSSPCICR[UInt(CRm<2:0>)];
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSSPCICR[UInt(CRm<2:0>)];
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSSPCICR[UInt(CRm<2:0>)];
  endif
endif

**MSR TRCSSPCICR<n>, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b0:n[2:0]</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
                else
                    TRCSSPCICR[UInt(CRm<2:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
                else
                    TRCSSPCICR[UInt(CRm<2:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
            else
                TRCSSPCICR[UInt(CRm<2:0>)] = X[t];

The TRCSTALLCTRL characteristics are:

**Purpose**

Enables trace unit functionality that prevents trace unit buffer overflows.

**Configuration**

AArch64 System register TRCSTALLCTRL bits [31:0] are architecturally mapped to External register TRCSTALLCTRL[31:0].

This register is present only when ETE is implemented and TRCIDR3.STALLCTL == 0b1. Otherwise, direct accesses to TRCSTALLCTRL are UNDEFINED.

**Attributes**

TRCSTALLCTRL is a 64-bit register.

**Field descriptions**

The TRCSTALLCTRL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>NOOVERFLOW, bit [13]</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [63:14]**

Reserved, RES0.

**NOOVERFLOW, bit [13]**

When TRCIDR3.NOOVERFLOW == 0b1:

Trace overflow prevention.

<table>
<thead>
<tr>
<th>NOOVERFLOW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace unit buffer overflow prevention is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace unit buffer overflow prevention is enabled.</td>
</tr>
</tbody>
</table>

Note that enabling this feature might cause a significant performance impact.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [12:9]**

Reserved, RES0.
ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

<table>
<thead>
<tr>
<th>ISTALL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit must not stall the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit can stall the PE.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from 0b0000 to 0b1111.

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Minimal invasion.</td>
</tr>
<tr>
<td></td>
<td>This setting has a greater risk of a trace unit buffer overflow.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Maximum invasion.</td>
</tr>
<tr>
<td></td>
<td>Reduced risk of a trace unit buffer overflow.</td>
</tr>
</tbody>
</table>

Note that for some implementations, invasion might occur at the minimal invasion level.

It is IMPLEMENTATION DEFINED whether some of the least significant bits are supported. Arm recommends that bits[3:2] are supported.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCSTALLCTRL

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCSTALLCTRL

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWR_EL2.TRC == '1' then
    AArch64.SystemAccess Trap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCSTALLCTRLR;
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCSTALLCTRLR;
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      return TRCSTALLCTRLR;
end if;

MSR TRCSTALLCTRLR, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWR_EL2.TRC == '1' then
    AArch64.SystemAccess Trap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCSTALLCTRLR = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCSTALLCTRLR = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCSTALLCTRLR = X[t];
end if;
TRCSTATR, Trace Status Register

The TRCSTATR characteristics are:

**Purpose**

Returns the trace unit status.

**Configuration**

AArch64 System register TRCSTATR bits [31:0] are architecturally mapped to External register TRCSTATR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCSTATR are UNDEFINED.

**Attributes**

TRCSTATR is a 64-bit register.

**Field descriptions**

The TRCSTATR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>PMSTABLE</td>
<td>Programmers’ model stable.</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>IDLE</td>
<td>Idle status.</td>
</tr>
</tbody>
</table>

**Bits [63:2]**

Reserved, RES0.

**PMSTABLE, bit [1]**

Programmers’ model stable.

<table>
<thead>
<tr>
<th>PMSTABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The programmers’ model is not stable.</td>
</tr>
<tr>
<td>0b1</td>
<td>The programmers’ model is stable.</td>
</tr>
</tbody>
</table>

This bit is UNKNOWN while the trace unit is enabled.

**IDLE, bit [0]**

Idle status.

<table>
<thead>
<tr>
<th>IDLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit is not idle.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit is idle.</td>
</tr>
</tbody>
</table>

**Accessing the TRCSTATR**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCSR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  else
    return TRCSTATR;
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  else
    return TRCSTATR;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x10);
  else
    return TRCSTATR;
  endif
else
  return TRCSTATR;
endif

TRCSTATR, Trace Status Register

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
TRCSYNCPR, Synchronization Period Register

The TRCSYNCPR characteristics are:

**Purpose**

Controls how often trace protocol synchronization requests occur.

**Configuration**

AArch64 System register TRCSYNCPR bits [31:0] are architecturally mapped to External register TRCSYNCPR[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCSYNCPR are **UNDEFINED**.

**Attributes**

TRCSYNCPR is a 64-bit register.

**Field descriptions**

The TRCSYNCPR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>6</td>
<td>PERIOD</td>
</tr>
<tr>
<td>0</td>
<td>PERIOD</td>
</tr>
</tbody>
</table>

**Bits [63:5]**

Reserved, RES0.

**PERIOD, bits [4:0]**

Defines the number of bytes of trace between each periodic trace protocol synchronization request.
<table>
<thead>
<tr>
<th>PERIOD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Trace protocol synchronization is disabled.</td>
</tr>
<tr>
<td>0b01000</td>
<td>Trace protocol synchronization request occurs after $2^8$ bytes of trace.</td>
</tr>
<tr>
<td>0b01001</td>
<td>Trace protocol synchronization request occurs after $2^9$ bytes of trace.</td>
</tr>
<tr>
<td>0b01010</td>
<td>Trace protocol synchronization request occurs after $2^{10}$ bytes of trace.</td>
</tr>
<tr>
<td>0b01011</td>
<td>Trace protocol synchronization request occurs after $2^{11}$ bytes of trace.</td>
</tr>
<tr>
<td>0b01100</td>
<td>Trace protocol synchronization request occurs after $2^{12}$ bytes of trace.</td>
</tr>
<tr>
<td>0b01101</td>
<td>Trace protocol synchronization request occurs after $2^{13}$ bytes of trace.</td>
</tr>
<tr>
<td>0b01110</td>
<td>Trace protocol synchronization request occurs after $2^{14}$ bytes of trace.</td>
</tr>
<tr>
<td>0b01111</td>
<td>Trace protocol synchronization request occurs after $2^{15}$ bytes of trace.</td>
</tr>
<tr>
<td>0b10000</td>
<td>Trace protocol synchronization request occurs after $2^{16}$ bytes of trace.</td>
</tr>
<tr>
<td>0b10001</td>
<td>Trace protocol synchronization request occurs after $2^{17}$ bytes of trace.</td>
</tr>
<tr>
<td>0b10010</td>
<td>Trace protocol synchronization request occurs after $2^{18}$ bytes of trace.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Trace protocol synchronization request occurs after $2^{19}$ bytes of trace.</td>
</tr>
<tr>
<td>0b10100</td>
<td>Trace protocol synchronization request occurs after $2^{20}$ bytes of trace.</td>
</tr>
</tbody>
</table>

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.
- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCSYNCR**

Must be programmed if TRCIDR3.SYNCR == 0b0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

```
MRS <Xt>, TRCSYNPR
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSYNCPR;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSYNCPR;
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCSYNCPR;
            end
    end
MSR TRCSYNCPR, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSYNCPR = X[t];
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSYNCPR = X[t];
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                TRCSYNCPR = X[t];
            end
TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

**Purpose**

Sets the trace ID for instruction trace.

**Configuration**

AArch64 System register TRCTRACEIDR bits [31:0] are architecturally mapped to External register TRCTRACEIDR[31:0].

This register is present only when ETE is implemented and TRCIDR5.TRACEIDSIZE != 0x00. Otherwise, direct accesses to TRCTRACEIDR are UNDEFINED.

**Attributes**

TRCTRACEIDR is a 64-bit register.

**Field descriptions**

The TRCTRACEIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Trace ID field, TRACEID</td>
</tr>
</tbody>
</table>

**Bits [63:7]**

Reserved, RES0.

**TRACEID, bits [6:0]**

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of TRCIDR5.TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCTRACEIDR**

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:
MRS <Xt>, TRCTRACEIDR

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCTRACEIDR;
    elsif PSTATE.EL == EL2 then
        if CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCTRACEIDR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCTRACEIDR = X[t];
else
    TRCTRACEIDR = X[t];

MSR TRCTRACEIDR, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCTRACEIDR = X[t];
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCTRACEIDR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCTRACEIDR = X[t];
TRCTSCTRL, Timestamp Control Register

The TRCTSCTRL characteristics are:

Purpose

Controls the insertion of global timestamps in the trace stream.

Configuration

AArch64 System register TRCTSCTRL bits [31:0] are architecturally mapped to External register TRCTSCTRL[31:0].

This register is present only when ETE is implemented and TRCId[0].TSSIZE != 0b0000. Otherwise, direct accesses to TRCTSCTRL are UNDEFINED.

Attributes

TRCTSCTRL is a 64-bit register.

Field descriptions

The TRCTSCTRL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | EVENT_TYPE | RES0 | EVENT_SEL |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

Bits [63:8]

Reserved, RES0.

EVENT_TYPE, bit [7]

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCTSCTRL EVENT_SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCTSCTRL EVENT_SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTRL EVENT_SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

This bit is RES0 if TRCId[4].NUMRSPAIR == 0b0000.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.
EVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTRL.EVENT.TYPE controls whether TRCTSCTRL.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

This field is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCTSCTRL

Must be programmed if TRCCONFIGR.TS == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS <Xt>, TRCTSCTRL

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elself HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCTSCTRL;
elseif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elself HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCTSCTRL;
elseif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
else
  return TRCTSCTRL;

MSR TRCTSCTRL, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    TRCTSCTRL = X[t];
  end
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCTSCTRL = X[t];
  end
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCTSCTRL = X[t];
  end
TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR characteristics are:

**Purpose**

Controls instruction trace filtering.

**Configuration**

AArch64 System register TRCVICTLR bits [31:0] are architecturally mapped to External register TRCVICTLR[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCVICTLR are UNDEFINED.

**Attributes**

TRCVICTLR is a 64-bit register.

**Field descriptions**

The TRCVICTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:23]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>EXLEVEL_NS_EL2, bit [22]</td>
<td>Filter instruction trace for EL2 in Non-secure state.</td>
</tr>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL2 in Non-secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL2 in Non-secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

EXLEVEL_NS_EL1, bit [21]  
When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.
EXLEVEL_NS_EL1, bit [20]

When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL1 in Non-secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL1 in Non-secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [20]

When Non-secure EL0 is implemented:

Filter instruction trace for EL0 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL0 in Non-secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL0 in Non-secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]

When HaveEL(EL3):

Filter instruction trace for EL3 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL3 in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL3 in Secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]

When HaveEL(EL2) and ARMv8.4-SecEL2 is implemented:

Filter instruction trace for EL2 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL2 in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL2 in Secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Reserved, RES0.

**EXLEVEL_S_EL1, bit [17]**

*When Secure EL1 is implemented:*

Filter instruction trace for EL1 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL1 in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL1 in Secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Reserved, RES0.

**EXLEVEL_S_EL0, bit [16]**

*When Secure EL0 is implemented:*

Filter instruction trace for EL0 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL0 in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL0 in Secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Reserved, RES0.

**Bits [15:12]**

Reserved, RES0.

**TRCERR, bit [11]**

*When TRCIDR3.TRCCERR == 0b1:*

Controls the forced tracing of System Error exceptions.

<table>
<thead>
<tr>
<th>TRCERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forced tracing of System Error exceptions is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forced tracing of System Error exceptions is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Reserved, RES0.
TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

<table>
<thead>
<tr>
<th>TRCRESET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forced tracing of PE Resets is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forced tracing of PE Resets is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

<table>
<thead>
<tr>
<th>SSSTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stopped State. The ViewInst start/stop function is in the stopped state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Started State. The ViewInst start/stop function is in the started state.</td>
</tr>
</tbody>
</table>

Before software enables the trace unit, it must write to this bit to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this bit to 0b1. Arm recommends that the value of this bit is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR.SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing TRCPBGCTRL.EN or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

This bit is res1 if TRCIDR4.NUMACPAIRS == 0b0000 and TRCIDR4.NUMPC == 0b0000.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCVICTLR.EVENT_SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT_SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT_SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
Bits [6:5]
Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == 0b0000:

This field is reserved:
• Bits [4:1] are RES0.
• Bit [0] is RES1.

Otherwise:
Reserved, RES0.

Accessing the TRCVICTLR

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings:

MRS <Xt>, TRCVICTLR

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGREWTR_EL2.TRCVICTLR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCVICTLR;
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCVICTLR;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    return TRCVICTLR;
  endif
endif

MSR TRCVICTLR, \langle X[t]\rangle

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGREWTR_EL2.TRCVICTLR == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVICTLR = X[t];
  endif
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVICTLR = X[t];
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVICTLR = X[t];
  endif
endif
The TRCVIIECTLR characteristics are:

**Purpose**

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

**Configuration**

AArch64 System register TRCVIIECTLR bits [31:0] are architecturally mapped to External register TRCVIIECTLR[31:0].

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVIIECTLR are UNDEFINED.

**Attributes**

TRCVIIECTLR is a 64-bit register.

**Field descriptions**

The TRCVIIECTLR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | EXCLUDE<m>, bit [m+16] | RES0 | INCLUDE<m>, bit [m] |

**Bits [63:24]**

Reserved, RES0.

**EXCLUDE<m>, bit [m+16], for m = 0 to 7**

Selects which Address Range Comparators are in use with the ViewInst exclude function.

Each bit represents an Address Range Comparator, so bit[m] controls the selection of Address Range Comparator m.

<table>
<thead>
<tr>
<th>EXCLUDE&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address range that Address Range Comparator m defines, is not selected for the ViewInst exclude function.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address range that Address Range Comparator m defines, is selected for the ViewInst exclude function.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [15:8]**

Reserved, RES0.
**INCLUDE<\textit{m}>, bit \[m\], for \textit{m} = 0 to 7**

Selects which Address Range Comparators are in use with the ViewInst include function.

Each bit represents an Address Range Comparator, so bit\([m]\) controls the selection of Address Range Comparator \(m\).

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

<table>
<thead>
<tr>
<th>INCLUDE&lt;\textit{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address range that Address Range Comparator (m) defines, is not selected for the ViewInst include function.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address range that Address Range Comparator (m) defines, is selected for the ViewInst include function.</td>
</tr>
</tbody>
</table>

This bit is \texttt{res0} if \(m \geq \text{TRCIDR4.NUMACPAIRS}\).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the TRCVIIECTLR**

Must be programmed if \texttt{TRCIDR4.NUMACPAIRS} > \texttt{0b0000}.

Writes are \texttt{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

MRS \(<\textit{Xt}>, \text{TRCVIIECTLR}\)

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

if \(\text{PSTATE.EL} == \text{EL0}\) then
  \texttt{UNDEFINED};
elsif \(\text{PSTATE.EL} == \text{EL1}\) then
  if \(\text{CPACR_EL1.TTA} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL1, 0x18)};
  elsif \(\text{EL2Enabled()} \&\& \!\text{ELUsingAArch32(EL2)} \&\& \text{CPTR_EL2.TTA} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
  elsif \(\text{EL2Enabled()} \&\& \!\text{ELUsingAArch32(EL2)} \&\& (!\text{HaveEL(EL3)} \&\& \text{SCR_EL3.FGTEn} == '1') \&\& \text{HDFGTR_EL2.TRC} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
  elsif \(\text{HaveEL(EL3)} \&\& \!\text{ELUsingAArch32(EL3)} \&\& \text{CPTR_EL3.TTA} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL3, 0x18)};
  else
    return TRCVIIECTLR;
elsif \(\text{PSTATE.EL} == \text{EL2}\) then
  if \(\text{CPTR_EL2.TTA} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL2, 0x18)};
  elsif \(\text{HaveEL(EL3)} \&\& \!\text{ELUsingAArch32(EL3)} \&\& \text{CPTR_EL3.TTA} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL3, 0x18)};
  else
    return TRCVIIECTLR;
elsif \(\text{PSTATE.EL} == \text{EL3}\) then
  if \(\text{CPTR_EL3.TTA} == '1'\) then
    \texttt{AArch64.SystemAccessTrap(EL3, 0x18)};
  else
    return TRCVIIECTLR;
else
  return TRCVIIECTLR;
```

MSR \texttt{TRCVIIECTLR, <\textit{Xt}>}

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
</table>
```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TRCVIIECTLR = X[t];
    end if;
else
    TRCVIIECTLR = X[t];
end if;
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVIIECTLR = X[t];
    end if;
else
    TRCVIIECTLR = X[t];
end if;
else
    TRCVIIECTLR = X[t];
end if;
TRCVIPCSSCTRL, ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTRL characteristics are:

**Purpose**

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

**Configuration**

AArch64 System register TRCVIPCSSCTRL bits [31:0] are architecturally mapped to External register TRCVIPCSSCTRL[31:0].

This register is present only when ETE is implemented and TRCIDR4.NUMPC > 0b0000. Otherwise, direct accesses to TRCVIPCSSCTRL are UNDEFINED.

**Attributes**

TRCVIPCSSCTRL is a 64-bit register.

**Field descriptions**

The TRCVIPCSSCTRL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | STOP<m>, bit [m+16] | RES0 | START<m>, bit [m] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:24]**

Reserved, RES0.

**STOP<m>, bit [m+16], for m = 0 to 7**

Selects which PE Comparator Inputs are in use with ViewInst start/stop function, for the purpose of stopping trace.

<table>
<thead>
<tr>
<th>STOP&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PE Comparator Input m, is not selected as a stop resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PE Comparator Input m, is selected as a stop resource.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR4.NUMPC.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [15:8]**

Reserved, RES0.

**START<m>, bit [m], for m = 0 to 7**

Selects which PE Comparator Inputs are in use with ViewInst start/stop function, for the purpose of starting trace.
<table>
<thead>
<tr>
<th>START(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PE Comparator Input m, is not selected as a start resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PE Comparator Input m, is selected as a start resource.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq \text{TRCIDR4.NUMPC}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

### Accessing the TRCVIPCSSCTRLR

Must be programmed if \(\text{TRCIDR4.NUMPC} \neq 0b0000\).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

\[
\begin{array}{cccccc}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b10 & 0b001 & 0b0000 & 0b0011 & 0b010 \\
\end{array}
\]

```c
if \(\text{PSTATE.EL} == \text{EL0} \) then
    \text{UNDEFINED};
elsif \(\text{PSTATE.EL} == \text{EL1} \) then
    if \(\text{CPACR_EL1.TTA} == '1' \) then
        \text{AArch64.SystemAccessTrap(EL1, 0x18)};
    elsif \(\text{EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& CPTR_EL2.TTA} == '1' \) then
        \text{AArch64.SystemAccessTrap(EL2, 0x18)};
    elsif \(\text{EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& (!HaveEL(EL3) \mid \text{SCR_EL3.FGTEn} == '1') \&\& \text{HDFGRTR_EL2.TRC} == '1' \) then
        \text{AArch64.SystemAccessTrap(EL2, 0x18)};
    else
        \text{return TRCVIPCSSCTRLR};
    end
elsif \(\text{PSTATE.EL} == \text{EL2} \) then
    if \(\text{CPTR_EL2.TTA} == '1' \) then
        \text{AArch64.SystemAccessTrap(EL2, 0x18)};
    elsif \(\text{HaveEL(EL3) \&\& !ELUsingAArch32(EL3) \&\& CPTR_EL3.TTA} == '1' \) then
        \text{AArch64.SystemAccessTrap(EL3, 0x18)};
    else
        \text{return TRCVIPCSSCTRLR};
    end
elsif \(\text{PSTATE.EL} == \text{EL3} \) then
    if \(\text{CPTR_EL3.TTA} == '1' \) then
        \text{AArch64.SystemAccessTrap(EL3, 0x18)};
    else
        \text{return TRCVIPCSSCTRLR};
    end
end
```

\[
\begin{array}{cccccc}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} \\
0b10 & 0b001 & 0b0000 & 0b0011 & 0b010 \\
\end{array}
\]
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL3Enabled() && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVIPCSSCTRL = X[t];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVIPCSSCTRL = X[t];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVIPCSSCTRL = X[t];
    end
TRCVISSCTLR, ViewInst Start/Stop Control Register

The TRCVISSCTL characteristics are:

**Purpose**

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

**Configuration**

AArch64 System register TRCVISSCTL[31:0] are architecturally mapped to External register TRCVISSCTL[31:0].

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVISSCTL are **UNDEFINED**.

**Attributes**

TRCVISSCTL is a 64-bit register.

**Field descriptions**

The TRCVISSCTL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| STOP<m>, bit [m+16] | RES0 | START<m>, bit [m] |

**Bits [63:32]**

Reserved, RES0.

**STOP<m>, bit [m+16], for m = 0 to 15**

Selects which Single Address Comparators are in use with ViewInst start/stop function, for the purpose of stopping trace.

<table>
<thead>
<tr>
<th>STOP&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single Address Comparator m, is not selected as a stop resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single Address Comparator m, is selected as a stop resource.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= 2 \times TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**START<m>, bit [m], for m = 0 to 15**

Selects which Single Address Comparators are in use with ViewInst start/stop function, for the purpose of starting trace.

<table>
<thead>
<tr>
<th>START&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single Address Comparator m, is not selected as a start resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single Address Comparator m, is selected as a start resource.</td>
</tr>
</tbody>
</table>
This bit is \texttt{res0} if \( m \geq 2 \times \text{TRCIDR4.NUMACPAIRS} \).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the TRCVISSCTLR**

Must be programmed if \texttt{TRCIDR4.NUMACPAIRS > 0b0000}.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are \texttt{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

\[
\begin{array}{cccccc}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b10 & 0b001 & 0b0000 & 0b0010 & 0b010
\end{array}
\]

if \( \text{PSTATE.EL} == \text{EL0} \) then
\( \text{UNDEFINED} \);
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
if \( \text{CPACR_EL1.TTA} == '1' \) then
\( \text{AArch64.SystemAccessTrap(EL1, 0x18)} \);
elsif \( \text{EL2Enabled()} \land \!\! \text{ELUsingAArch32(EL2)} \land \!\! \text{CPTR_EL2.TTA} == '1' \) then
\( \text{AArch64.SystemAccessTrap(EL2, 0x18)} \);
elsif \( \text{EL2Enabled()} \land \!\! \text{ELUsingAArch32(EL2)} \land (\!\! \! \text{HaveEL(EL3)} \lor \!\! \text{SCR_EL3.FGTEn == '1'} \land \!\! \text{HDFGRTR_EL2.TRC} == '1') \) then
\( \text{AArch64.SystemAccessTrap(EL2, 0x18)} \);
elsif \( \text{HaveEL(EL3)} \land \!\! \text{ELUsingAArch32(EL3)} \land \!\! \text{CPTR_EL3.TTA} == '1' \) then
\( \text{AArch64.SystemAccessTrap(EL3, 0x18)} \);
else
return \text{TRCVISSCTLR};
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
if \( \text{CPTR_EL2.TTA} == '1' \) then
\( \text{AArch64.SystemAccessTrap(EL2, 0x18)} \);
elsif \( \text{HaveEL(EL3)} \land \!\! \text{ELUsingAArch32(EL3)} \land \!\! \text{CPTR_EL3.TTA} == '1' \) then
\( \text{AArch64.SystemAccessTrap(EL3, 0x18)} \);
else
return \text{TRCVISSCTLR};
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
if \( \text{CPTR_EL3.TTA} == '1' \) then
\( \text{AArch64.SystemAccessTrap(EL3, 0x18)} \);
else
return \text{TRCVISSCTLR};
\end{array}
\]

\[
\begin{array}{cccccc}
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
0b10 & 0b001 & 0b0000 & 0b0010 & 0b010
\end{array}
\]
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVISSCTLR = X[t];
    end if
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVISSCTLR = X[t];
    end if
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVISSCTLR = X[t];
    end if
else
    TRCVISSCTLR = X[t];
end if

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TRCVMIDCCTLR0, Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCCTLR0 characteristics are:

**Purpose**

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<\text{n}> registers, where \( n=0-3 \).

**Configuration**

AArch64 System register TRCVMIDCCTLR0 bits [31:0] are architecturally mapped to External register TRCVMIDCCTLR0[31:0].

This register is present only when ETE is implemented, TRCIDR4.NUMVMIDC > 0x0 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTLR0 are UNDEFINED.

**Attributes**

TRCVMIDCCTLR0 is a 64-bit register.

**Field descriptions**

The TRCVMIDCCTLR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP3&lt;m&gt;, bit [m+24]</td>
<td>The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>COMP2&lt;m&gt;, bit [m+16]</td>
<td>The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>COMP1&lt;m&gt;, bit [m+8]</td>
<td>The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>COMP0&lt;m&gt;, bit [m]</td>
<td>The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

COMP3<m>, bit [m+24], for \( m = 0 \) to 7

When TRCIDR4.NUMVMIDC > 3:

TRCVMIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR3. Each bit in this field corresponds to a byte in TRCVMIDCVR3.

<table>
<thead>
<tr>
<th>COMP3&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \( m \geq \text{TRCIDR2.VMIDSIZE} \).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
COMP2< m >, bit [ m+16], for m = 0 to 7

When TRCIDR4.NUMVMIDC > 2:

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

<table>
<thead>
<tr>
<th>COMP2&lt; m &gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if m >= TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP1< m >, bit [ m+8], for m = 0 to 7

When TRCIDR4.NUMVMIDC > 1:

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

<table>
<thead>
<tr>
<th>COMP1&lt; m &gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if m >= TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP0< m >, bit [ m], for m = 0 to 7

When TRCIDR4.NUMVMIDC > 0:

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

<table>
<thead>
<tr>
<th>COMP0&lt; m &gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if m >= TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RES0.

**Accessing the TRCVMIDCCTLR0**

If software uses the TRCVMIDCVR\textsubscript{n} registers, where n=0-3, then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCVMIDCVR\textsubscript{n} to 0x00.

If any bit is 0b1 and the relevant byte in TRCVMIDCVR\textsubscript{n} is not 0x00, the behavior of the Virtual Context Identifier Comparator is **CONSTRAINED UNPREDICTABLE**. In this scenario the comparator might match unexpectedly or might not match.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

\[
\text{MRS <Xt>, TRCVMIDCCTLR0}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCVMIDCCTLR0;
    end if;
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVMIDCCTLR0;
    end if;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVMIDCCTLR0;
    end if;
else
    return TRCVMIDCCTLR0;
end if;

\[
\text{MSR TRCVMIDCCTLR0, <Xt>}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVMIDCCTLR0 = X[t];
  elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCVMIDCCTLR0 = X[t];
  elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
      AArch64.SystemAccessTrap(EL3, 0x18);
    else
      TRCVMIDCCTLR0 = X[t];

The TRCVMIDCCTRL1 characteristics are:

**Purpose**

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<\text{n}> registers, where \text{n}=4-7.

**Configuration**

AArch64 System register TRCVMIDCCTRL1 bits [31:0] are architecturally mapped to External register TRCVMIDCCTRL1[31:0].

This register is present only when ETE is implemented, TRCIDR4.NUMVMIDC > 0x4 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTRL1 are **UNDEFINED**.

**Attributes**

TRCVMIDCCTRL1 is a 64-bit register.

**Field descriptions**

The TRCVMIDCCTRL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field Assignment</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:32]</td>
<td><strong>RES0</strong></td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>[31:24]</td>
<td>COMP7\text{&lt;m&gt;}, bit \text{[m+24]}</td>
<td>When TRCIDR4.NUMVMIDC &gt; 7:</td>
</tr>
<tr>
<td>[23:16]</td>
<td>COMP6\text{&lt;m&gt;}, bit \text{[m+16]}</td>
<td></td>
</tr>
<tr>
<td>[15:8]</td>
<td>COMP5\text{&lt;m&gt;}, bit \text{[m+8]}</td>
<td></td>
</tr>
<tr>
<td>[7:0]</td>
<td>COMP4\text{&lt;m&gt;}, bit \text{[m]}</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**COMP7\text{<m>}, bit \text{[m+24]}, for \text{m} = 0 to 7**

When TRCIDR4.NUMVMIDC > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

<table>
<thead>
<tr>
<th>COMP7\text{&lt;m&gt;}</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if \text{m} >= TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.
COMP6<\(m\)>, bit \([m+16]\), for \(m = 0\) to \(7\)

When TRCIDR4.NUMVMIDC > 6:

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

<table>
<thead>
<tr>
<th>COMP6&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR6([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR6([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq\) TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, res0.

COMP5<\(m\)>, bit \([m+8]\), for \(m = 0\) to \(7\)

When TRCIDR4.NUMVMIDC > 5:

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

<table>
<thead>
<tr>
<th>COMP5&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR5([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR5([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq\) TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, res0.

COMP4<\(m\)>, bit \([m]\), for \(m = 0\) to \(7\)

When TRCIDR4.NUMVMIDC > 4:

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

<table>
<thead>
<tr>
<th>COMP4&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR4([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR4([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq\) TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RES0.

Accessing the TRCVMIDCCTL_R1

If software uses the TRCVMIDCVR<\text{n}> registers, where \(n=4-7\), then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCVMIDCVR<\text{n}> to 0x00.

If any bit is 0b1 and the relevant byte in TRCVMIDCVR<\text{n}> is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings:

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b10 & 0b001 & 0b0011 & 0b0011 & 0b010 \\
\hline
\end{array}
\]

if PSTATE.EL == EL0 then
UNDEFINED;
elsf if PSTATE.EL == EL1 then
if CPACR_EL1.TTA == '1' then
AArch64.SystemAccessTrap(EL1, 0x18);
e1sif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
e1sif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.TRC == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
e1sif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
AArch64.SystemAccessTrap(EL3, 0x18);
else
return TRCVMIDCCTL_R1;
elsf PSTATE.EL == EL2 then
if CPTR_EL2.TTA == '1' then
AArch64.SystemAccessTrap(EL2, 0x18);
e1sif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
AArch64.SystemAccessTrap(EL3, 0x18);
else
return TRCVMIDCCTL_R1;
elsf PSTATE.EL == EL3 then
if CPTR_EL3.TTA == '1' then
AArch64.SystemAccessTrap(EL3, 0x18);
else
return TRCVMIDCCTL_R1;

MSR TRCVMIDCCTL_R1, <Xt>

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{op0} & \text{op1} & \text{CRn} & \text{CRm} & \text{op2} \\
\hline
0b10 & 0b001 & 0b0011 & 0b0011 & 0b010 \\
\hline
\end{array}
\]
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.TTA == '1' then
    AArch64.SystemAccessTrap(EL1, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVMIDCCTLR1 = X[t];
  end
elsif PSTATE.EL == EL2 then
  if CPTR_EL2.TTA == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVMIDCCTLR1 = X[t];
  end
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.TTA == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRCVMIDCCTLR1 = X[t];
end
TRCVMIDCVR<n>, Virtual Context Identifier Comparator Value Register <n>, n = 0 - 7

The TRCVMIDCVR<n> characteristics are:

** Purpose**

Contains the Virtual Context Identifier Comparator value.

** Configuration**

AArch64 System register TRCVMIDCVR<n> bits [63:0] are architecturally mapped to External register TRCVMIDCVR<n>[63:0].

This register is present only when ETE is implemented and TRCIDR4.NUMVMIDC > n. Otherwise, direct accesses to TRCVMIDCVR<n> are UNDEFINED.

** Attributes**

TRCVMIDCVR<n> is a 64-bit register.

** Field descriptions**

The TRCVMIDCVR<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VALUE | VALUE |

** VALUE, bits [63:0]**

Virtual context identifier value. The width of this field is indicated by TRCIDR2.VMIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

** Accessing the TRCVMIDCVR<n>**

Must be programmed if any of the following are true:

- TRCRSCTLR<a>.GROUP == 0b0111 and TRCRSCTLR<a>.VMID[n] == 0b1.
- TRCACATR<a>.CONTEXTTYPE == 0b10 or 0b11 and TRCACATR<a>.CONTEXT == n.

Accesses to this register use the following encodings:

MRS <Xt>, TRCVMIDCVR<n>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>n[2:0]:0b0</td>
<td>0b001</td>
</tr>
</tbody>
</table>

Page 1945
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCVMIDCVR[UInt(CRm<3:1>)];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];
    end
end

MSR TRCVMIDCVR<n>, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>n[2:0]:0b0</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRCVMIDCVR[UInt(CRm<3:1>)];
    end
elsif PSTATE.EL == EL2 then
    if CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];
    end
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];
    end
TRFCR_EL1, Trace Filter Control Register (EL1)

The TRFCR_EL1 characteristics are:

**Purpose**

Provides EL1 controls for Trace.

**Configuration**

AArch64 System register TRFCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register TRFCR[31:0].

This register is present only when ARMv8.4-Trace is implemented. Otherwise, direct accesses to TRFCR_EL1 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TRFCR_EL1 is a 64-bit register.

**Field descriptions**

The TRFCR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | TS | RES0 | E1TRE | E0TRE |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:7]**

Reserved, RES0.

**TS, bits [6:5]**

Timestamp Control

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>Virtual timestamp. The traced timestamp is the physical counter value, minus the value of CNTVOFF_EL2.</td>
<td>When ARMv8.6-ECV is implemented</td>
</tr>
<tr>
<td>0b10</td>
<td>Guest Physical timestamp. The traced timestamp is the physical counter value, minus the value of CNTPOFF_EL2.</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>Physical timestamp. The traced timestamp is the physical counter value.</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and TRFCR_EL2.TS != 0b00.
When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:

- `CNTHCTL_EL2.ECV` is 0.
- `SCR_EL3.ECVEn` is 0.
- `HCR_EL2.{E2H, TGE}` is \{1, 1\}.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [4:2]**

Reserved, RES0.

**E1TRE, bit [1]**

EL1 Trace Enable.

<table>
<thead>
<tr>
<th>E1TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace is prohibited at EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace is allowed at EL1.</td>
</tr>
</tbody>
</table>

This field is ignored if `SelfHostedTraceEnabled() == FALSE`.

On a Warm reset, this field resets to 0.

**E0TRE, bit [0]**

EL0 Trace Enable.

<table>
<thead>
<tr>
<th>E0TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace is prohibited at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace is allowed at EL0.</td>
</tr>
</tbody>
</table>

This field is ignored if any of the following are true:

- `SelfHostedTraceEnabled() == FALSE`.
- EL2 is implemented and enabled in the current Security state and `HCR_EL2.TGE == 1`.

On a Warm reset, this field resets to 0.

**Accessing the TRFCR_EL1**

Accesses to this register use the following encodings:

MRS <Xt>, TRFCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x880];
    else
        return TRFCR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return TRFCR_EL2;
        else
            return TRFCR_EL1;
    elsif PSTATE.EL == EL3 then
        return TRFCR_EL1;

MSR TRFCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HDFGWTR_EL2.TRFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x880] = X[t];
    else
        TRFCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            TRFCR_EL2 = X[t];
        else
            TRFCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        TRFCR_EL1 = X[t];

MRS <Xt>, TRFCR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x880];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRFCR_EL1 = X[t];
        end
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TRFCR_EL1 = X[t];
    else
        UNDEFINED;

MSR TRFCR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b001</td>
<td>0b001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x880] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRFCR_EL1 = X[t];
        end
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TRFCR_EL1 = X[t];
    else
        UNDEFINED;
TRFCR_EL2, Trace Filter Control Register (EL2)

The TRFCR_EL2 characteristics are:

**Purpose**

Provides EL2 controls for Trace.

**Configuration**

AArch64 System register TRFCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HTRFCR[31:0].

This register is present only when ARMv8.4-Trace is implemented. Otherwise, direct accesses to TRFCR_EL2 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch64. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TRFCR_EL2 is a 64-bit register.

**Field descriptions**

The TRFCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>61</td>
<td>TS</td>
</tr>
<tr>
<td>60</td>
<td>RES0</td>
</tr>
<tr>
<td>59</td>
<td>RES0</td>
</tr>
<tr>
<td>58</td>
<td>RES0</td>
</tr>
<tr>
<td>57</td>
<td>E2TRE</td>
</tr>
<tr>
<td>56</td>
<td>RES0</td>
</tr>
<tr>
<td>55</td>
<td>RES0</td>
</tr>
<tr>
<td>54</td>
<td>E0HTRE</td>
</tr>
<tr>
<td>53</td>
<td>RES0</td>
</tr>
<tr>
<td>52</td>
<td>RES0</td>
</tr>
<tr>
<td>51</td>
<td>RES0</td>
</tr>
<tr>
<td>50</td>
<td>RES0</td>
</tr>
<tr>
<td>49</td>
<td>RES0</td>
</tr>
<tr>
<td>48</td>
<td>RES0</td>
</tr>
<tr>
<td>47</td>
<td>RES0</td>
</tr>
<tr>
<td>46</td>
<td>RES0</td>
</tr>
<tr>
<td>45</td>
<td>RES0</td>
</tr>
<tr>
<td>44</td>
<td>RES0</td>
</tr>
<tr>
<td>43</td>
<td>RES0</td>
</tr>
<tr>
<td>42</td>
<td>RES0</td>
</tr>
<tr>
<td>41</td>
<td>RES0</td>
</tr>
<tr>
<td>40</td>
<td>RES0</td>
</tr>
<tr>
<td>39</td>
<td>RES0</td>
</tr>
<tr>
<td>38</td>
<td>RES0</td>
</tr>
<tr>
<td>37</td>
<td>RES0</td>
</tr>
<tr>
<td>36</td>
<td>RES0</td>
</tr>
<tr>
<td>35</td>
<td>RES0</td>
</tr>
<tr>
<td>34</td>
<td>RES0</td>
</tr>
<tr>
<td>33</td>
<td>RES0</td>
</tr>
<tr>
<td>32</td>
<td>RES0</td>
</tr>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>RES0</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>RES0</td>
</tr>
<tr>
<td>25</td>
<td>RES0</td>
</tr>
<tr>
<td>24</td>
<td>RES0</td>
</tr>
<tr>
<td>23</td>
<td>RES0</td>
</tr>
<tr>
<td>22</td>
<td>RES0</td>
</tr>
<tr>
<td>21</td>
<td>RES0</td>
</tr>
<tr>
<td>20</td>
<td>RES0</td>
</tr>
<tr>
<td>19</td>
<td>RES0</td>
</tr>
<tr>
<td>18</td>
<td>RES0</td>
</tr>
<tr>
<td>17</td>
<td>RES0</td>
</tr>
<tr>
<td>16</td>
<td>RES0</td>
</tr>
<tr>
<td>15</td>
<td>RES0</td>
</tr>
<tr>
<td>14</td>
<td>RES0</td>
</tr>
<tr>
<td>13</td>
<td>RES0</td>
</tr>
<tr>
<td>12</td>
<td>RES0</td>
</tr>
<tr>
<td>11</td>
<td>RES0</td>
</tr>
<tr>
<td>10</td>
<td>RES0</td>
</tr>
<tr>
<td>9</td>
<td>RES0</td>
</tr>
<tr>
<td>8</td>
<td>RES0</td>
</tr>
<tr>
<td>7</td>
<td>RES0</td>
</tr>
<tr>
<td>6</td>
<td>RES0</td>
</tr>
<tr>
<td>5</td>
<td>RES0</td>
</tr>
<tr>
<td>4</td>
<td>RES0</td>
</tr>
<tr>
<td>3</td>
<td>RES0</td>
</tr>
<tr>
<td>2</td>
<td>RES0</td>
</tr>
<tr>
<td>1</td>
<td>RES0</td>
</tr>
<tr>
<td>0</td>
<td>RES0</td>
</tr>
</tbody>
</table>

**Bits [63:7]**

Reserved, RES0.

**TS, bits [6:5]**

Timestamp Control. Controls which timebase is used for trace timestamps.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Timestamp controlled by TRFCR_EL1.TS or TRFCR.TS.</td>
<td></td>
</tr>
<tr>
<td>0b01</td>
<td>Virtual timestamp. The traced timestamp is the physical counter value, minus the value of CNTVOFF_EL2.</td>
<td>When ARMv8.6-ECV is implemented</td>
</tr>
<tr>
<td>0b10</td>
<td>Guest Physical timestamp. The traced timestamp is the physical counter value, minus the value of CNTPOFF_EL2.</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>Physical timestamp. The traced timestamp is the physical counter value.</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is ignored if SelfHostedTraceEnabled() == FALSE.
When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:

- \texttt{CNTHCTL\_EL2.ECV} is 0.
- \texttt{SCR\_EL3.ECVEn} is 0.
- \texttt{HCR\_EL2.(E2H, TGE)} is \{1, 1\}.

On a Warm reset, this field resets to 0.

**Bit [4]**

Reserved, \texttt{RES0}.

**CX, bit [3]**

\texttt{CONTEXTIDR\_EL2} and VMID trace enable.

<table>
<thead>
<tr>
<th>CX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>\texttt{CONTEXTIDR_EL2} and VMID trace prohibited.</td>
</tr>
<tr>
<td>0b1</td>
<td>\texttt{CONTEXTIDR_EL2} and VMID trace allowed.</td>
</tr>
</tbody>
</table>

This field is ignored if \texttt{SelfHostedTraceEnabled()} == FALSE.

On a Warm reset, this field resets to 0.

**Bit [2]**

Reserved, \texttt{RES0}.

**E2TRE, bit [1]**

EL2 Trace Enable.

<table>
<thead>
<tr>
<th>E2TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace is prohibited at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace is allowed at EL2.</td>
</tr>
</tbody>
</table>

This field is ignored if \texttt{SelfHostedTraceEnabled()} == FALSE.

On a Warm reset, this field resets to 0.

**E0HTRE, bit [0]**

EL0 Trace Enable.

<table>
<thead>
<tr>
<th>E0HTRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace is prohibited at EL0 when \texttt{HCR_EL2.TGE} == 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace is allowed at EL0 when \texttt{HCR_EL2.TGE} == 1.</td>
</tr>
</tbody>
</table>

This field is ignored if any of the following are true:

- \texttt{SelfHostedTraceEnabled()} == FALSE.
- EL2 is disabled in the current security state.
- \texttt{HCR\_EL2.TGE} == 0.

On a Warm reset, this field resets to 0.

**Accessing the TRFCR\_EL2**

Accesses to this register use the following encodings:
MRS <Xt>, TRFCR_EL2

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRFCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  return TRFCR_EL2;
endif

MRS TRFCR_EL2, <Xt>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRFCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  TRFCR_EL2 = X[t];
endif

MRS <Xt>, TRFCR_EL1

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
    AArch64.SystemAccessTrap(EL3, 0x18);
  else
    TRFCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  TRFCR_EL2 = X[t];
endif
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x880];
    else
        return TRFCR_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return TRFCR_EL2;
        else
            return TRFCR_EL1;
    elsif PSTATE.EL == EL3 then
        return TRFCR_EL1;
else
    MSR TRFCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b01</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRFCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x880] = X[t];
    else
        TRFCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            TRFCR_EL2 = X[t];
        else
            TRFCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        TRFCR_EL1 = X[t];
TTBR0_EL1, Translation Table Base Register 0 (EL1)

The TTBR0_EL1 characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL1&0 translation regime, and other information for this translation regime.

**Configuration**

AArch64 System register TTBR0_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR0[63:0]. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TTBR0_EL1 is a 64-bit register.

**Field descriptions**

The TTBR0_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ASID | BADDR | BADDR | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**ASID, bits [63:48]**

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

This field resets to an architecturally UNKNOWN value.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x], bits[47:1].

**Note**

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of TCR_EL1.IPS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If \( x >= 6 \) then \( z=x \).
  - Otherwise, \( z=6 \).
- When \( z>x \) register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When \( x>6 \) register bits[(x-1):6] are RES0.
Register bit[1] is RES0.

Bits[5:2] of the stage 1 translation table base address are zero.

In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.

### Note

- In an implementation that includes ARMv8.2-LPA a TCR_EL1 IPS value of 0b110, that selects an IPA size of 52 bits, is permitted only when using the 64KB translation granule.
- When the value of ID_AAA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of TCR_EL1 IPS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of TCR_EL1 IPS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

### Note

This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR0_EL1[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR0_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T0SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

### CnP, bit [0]

**When ARMv8.2-TTCNP is implemented:**

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0  | The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by:  
  - The value of TTBR0_EL1.CnP on those other PEs.  
  - The value of the current ASID.  
  - If EL2 is implemented and enabled in the current Security state, the value of the current VMID.  |
| 0b1  | The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply:  
  - The translation table entries are pointed to by TTBR0_EL1.  
  - The translation tables relate to the same translation regime.  
  - The ASID is the same as the current ASID.  
  - If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |
This field is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

**Note**

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Accessing the TTBR0_EL1**

When **HCR_EL2.E2H** is 1, without explicit synchronization, access from EL3 using the mnemonic TTBR0_EL1 or TTBR0_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, TTBR0_EL1**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRT_EL2.TTBR0_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x200];
  else
    return TTBR0_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return TTBR0_EL2;
  else
    return TTBR0_EL1;
elsif PSTATE.EL == EL3 then
  return TTBR0_EL1;

**MSR TTBR0_EL1, <Xt>**

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
      HFGWTR_EL2.TTBR0_EL1 == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
      TTBR0_EL1 = X[t];
   else
      TTBR0_EL1 = X[t];
   elsif PSTATE.EL == EL2 then
      if HCR_EL2.E2H == '1' then
         TTBR0_EL2 = X[t];
      else
         TTBR0_EL1 = X[t];
      elsif PSTATE.EL == EL3 then
         TTBR0_EL1 = X[t];
      else
         UNDEFINED;
   else
      UNDEFINED;
else
   if PSTATE.EL == EL0 then
      UNDEFINED;
else
   if PSTATE.EL == EL1 then
      if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
         return NVMem[0x200];
      elsif EL2Enabled() && HCR_EL2.NV == '1' then
         AArch64.SystemAccessTrap(EL2, 0x18); 
      else
         UNDEFINED;
   else
      UNDEFINED;
else
   if PSTATE.EL == EL2 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
         return TTBR0_EL1;
      else
         UNDEFINED;
   else
      UNDEFINED;
else
   if PSTATE.EL == EL3 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
         return TTBR0_EL1;
      else
         UNDEFINED;

MSR TTBR0_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
      return NVMem[0x200];
   elsif EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18); 
   else
      UNDEFINED;
else
   if PSTATE.EL == EL2 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
         return TTBR0_EL1;
      else
         UNDEFINED;
   else
      UNDEFINED;
else
   if PSTATE.EL == EL3 then
      if EL2Enabled() && HCR_EL2.E2H == '1' then
         return TTBR0_EL1;
      else
         UNDEFINED;

MSR TTBR0_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x200] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TTBR0_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TTBR0_EL1 = X[t];
    else
        UNDEFINED;
TTBR0_EL2, Translation Table Base Register 0 (EL2)

The TTBR0_EL2 characteristics are:

**Purpose**

When **HCR_EL2.E2H** is 0, holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

When **HCR_EL2.E2H** is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL2&0 translation regime, and other information for this translation regime.

**Configuration**

AArch64 System register TTBR0_EL2 bits [47:1] are architecturally mapped to AArch32 System register **HTTBR[47:1]**.

If EL2 is not implemented, this register is **RES0** from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

TTBR0_EL2 is a 64-bit register.

**Field descriptions**

The TTBR0_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASID</td>
<td>BADDR</td>
<td>CnP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

**ASID, bits [63:48]**

When ARMv8.1-VHE is implemented:

When **HCR_EL2.E2H** is 0, this field is **RES0**.

When **HCR_EL2.E2H** is 1, it holds an ASID for the translation table base address. The **TCR_EL2.A1** field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are **RES0**.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x], bits[47:1].
Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.

A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of \( \text{TCR} \_\text{EL2} \).\{1\}PS is \( 0b110 \), then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If \( x \geq 6 \) then \( z=x \).
  - Otherwise, \( z=6 \).
- When \( z>x \) register bits\((z-1):x\) are RES0, and bits\((z-1):x\) of the translation table base address are zero.
- When \( x>6 \) register bits\((x-1):6\) are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.
- In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.

Note

In an implementation that includes ARMv8.2-LPA:

- A \( \text{TCR} \_\text{EL2} \).\{1\}PS value of \( 0b110 \), that selects an OA size of 52 bits, is permitted only when using the 64KB translation granule.
- The OA size is specified by:
  - The value of \( \text{TCR} \_\text{EL2} \).PS when the value of \( \text{HCR} \_\text{EL2} \).E2H is 0.
  - The value of \( \text{TCR} \_\text{EL2} \).IPS when the value of \( \text{HCR} \_\text{EL2} \).E2H is 1.

When the value of \( \text{ID} \_\text{AA64MMFR0} \_\text{EL1} \).\text{PARange} indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of \( \text{TCR} \_\text{EL2} \).\{1\}PS is \( 0b110 \) and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of \( \text{TCR} \_\text{EL2} \).\{1\}PS is not \( 0b110 \) then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits\((x-1):1\) are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are \( 0b0000 \).

Note

This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR0 \_EL2[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR0 \_EL2, then the translation table base address might be misaligned, with effects that are \textsc{constrained unpredictable}, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how \( x \) is calculated based on the value of \( \text{TCR} \_\text{EL2} \).T0SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally \textsc{unknown} value.

CnP, bit [0]
When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by TTBR0_EL2 for the current translation regime, and ASID if applicable, are permitted to differ from corresponding entries for TTBR0_EL2 for other PEs in the Inner Shareable domain. This is not affected by:</td>
</tr>
<tr>
<td></td>
<td>• The value of TTBR0_EL2.CnP on those other PEs.</td>
</tr>
<tr>
<td></td>
<td>• When the current translation regime is the EL2&amp;0 regime, the value of the current ASID.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by TTBR0_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1 and all of the following apply:</td>
</tr>
<tr>
<td></td>
<td>• The translation table entries are pointed to by TTBR0_EL2.</td>
</tr>
<tr>
<td></td>
<td>• The translation tables relate to the same translation regime.</td>
</tr>
<tr>
<td></td>
<td>• If that translation regime is the EL2&amp;0 regime, the ASID is the same as the current ASID.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing the TTBR0_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic TTBR0_EL2 or TTBR0_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

MRS <Xt>, TTBR0_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return TTBR0_EL2;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL2;

MSR TTBR0_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.TTBR0_EL1 == '1' then
        return NVMem[0x200];
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR0_EL2;
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL1;

MRS <Xt>, TTBR0_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

MSR TTBR0_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
  UNDEFINED;
elsif PSTATE_EL == EL1 then
  if EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TTBR0_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x200] = X[t];
  else
    TTBR0_EL1 = X[t];
  end if;
elsif PSTATE_EL == EL2 then
  if HCR_EL2.E2H == '1' then
    TTBR0_EL2 = X[t];
  else
    TTBR0_EL1 = X[t];
  end if;
elsif PSTATE_EL == EL3 then
  TTBR0_EL1 = X[t];
end if;
TTBR0_EL3, Translation Table Base Register 0 (EL3)

The TTBR0_EL3 characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL3 translation regime, and other information for this translation regime.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to TTBR0_EL3 are *undefined*. RW fields in this register reset to architecturally *unknown* values.

**Attributes**

TTBR0_EL3 is a 64-bit register.

**Field descriptions**

The TTBR0_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>BADDR</td>
<td>BADDR</td>
<td>CnP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x].

**Note**

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of TCR_EL3.PS is 0b110 then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- When z>=x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.
- In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.
Note

- In an implementation that includes ARMv8.2-LPA a TCR_EL3 PS value of 0b110, that selects a PA size of 52 bits, is permitted only when using the 64KB translation granularity.
- When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52-bit PA size, if a translation table lookup uses this register with the 64KB translation granularity when the value of TCR_EL3 PS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of TCR_EL3 PS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

Note

This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granularity smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR0_EL3[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR0_EL3, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL3.T0SZ, the stage of translation, and the translation granularity size.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 for other PEs in the Inner Shareable domain. This is not affected by the value of TTBR0_EL3.CnP on those other PEs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1 and the translation table entries are pointed to by TTBR0_EL3.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries the results of translations using TTBR0_EL3 are CONSTRAINED UNPREDICTABLE, see ‘CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, `RES0`.

**Accessing the TTBR0_EL3**

Accesses to this register use the following encodings:

**MRS <Xt>, TTBR0_EL3**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if `PSTATE.EL == EL0` then
    UNDEFINED;
elsif `PSTATE.EL == EL1` then
    UNDEFINED;
elsif `PSTATE.EL == EL2` then
    UNDEFINED;
elsif `PSTATE.EL == EL3` then
    return TTBR0_EL3;

**MSR TTBR0_EL3, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if `PSTATE.EL == EL0` then
    UNDEFINED;
elsif `PSTATE.EL == EL1` then
    UNDEFINED;
elsif `PSTATE.EL == EL2` then
    UNDEFINED;
elsif `PSTATE.EL == EL3` then
    TTBR0_EL3 = X[t];
**TTBR1_EL1, Translation Table Base Register 1 (EL1)**

The TTBR1_EL1 characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL1&0 stage 1 translation regime, and other information for this translation regime.

**Configuration**

AArch64 System register TTBR1_EL1 bits [63:0] are architecturally mapped to AArch32 System register TTBR1[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TTBR1_EL1 is a 64-bit register.

**Field descriptions**

The TTBR1_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ASID | BADDR | BADDR | CnP |

**ASID, bits [63:48]**

An ASID for the translation table base address. The TCR_EL1.A1 field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0. This field resets to an architecturally UNKNOWN value.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x], bits[47:1].

**Note**

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of TCR_EL1.IPS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- When z>x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
Register bit[1] is RES0.
Bits[5:2] of the stage 1 translation table base address are zero.
In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.

Note

- In an implementation that includes ARMv8.2-LPA a TCR_EL1 IPS value of 0b110, that selects an IPA size of 52 bits, is permitted only when using the 64KB translation granule.
- When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of TCR_EL1 IPS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of TCR_EL1 IPS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

Note

This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR1_EL1[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR1_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL1.T1SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0 | The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by:
  - The value of TTBR1_EL1.CnP on those other PEs.
  - The value of the current ASID.
  - If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |
| 0b1 | The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply:
  - The translation table entries are pointed to by TTBR1_EL1.
  - The translation tables relate to the same translation regime.
  - The ASID is the same as the current ASID.
  - If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |
This field is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

---

**Note**

If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

---

**Otherwise:**

Reserved, RES0.

**Accessing the TTBR1_EL1**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic TTBR1_EL1 or TTBR1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, TTBR1_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() & HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() & HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() & HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR1_EL2;
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR1_EL1;

**MSR TTBR1_EL1, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TTBR1_EL1 == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x210] = X[t];
  else
    TTBR1_EL1 = X[t];
  elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
      TTBR1_EL2 = X[t];
    else
      TTBR1_EL1 = X[t];
  elsif PSTATE.EL == EL3 then
    TTBR1_EL1 = X[t];

MRS <Xt>, TTBR1_EL12

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x210];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return TTBR1_EL1;
  else
    UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    return TTBR1_EL1;
  else
    UNDEFINED;

MSR TTBR1_EL12, <Xt>

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x210] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TTBR1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        TTBR1_EL1 = X[t];
    else
        UNDEFINED;
TTBR1_EL2, Translation Table Base Register 1 (EL2)

The TTBR1_EL2 characteristics are:

**Purpose**

When HCR_EL2.E2H is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL2&0 translation regime, and other information for this translation regime.

**Note**

When HCR_EL2.E2H is 0, the contents of this register are ignored by the PE, except for a direct read or write of the register.

**Configuration**

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to TTBR1_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TTBR1_EL2 is a 64-bit register.

**Field descriptions**

The TTBR1_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>ASID</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>BADDR</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>CnP</td>
</tr>
</tbody>
</table>

**ASID, bits [63:48]**

An ASID for the translation table base address. The TCR_EL2.A1 field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

This field resets to an architecturally UNKNOWN value.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x], bits[47:1].

**Note**

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the
In an implementation that includes ARMv8.2-LPA, if the value of TCR_EL2.IPS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- When z>x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.
- In an implementation that includes ARMv8.2-TTCNP bit[0] of the stage 1 translation table base address is zero.

**Note**

- In an implementation that includes ARMv8.2-LPA a TCR_EL2.IPS value of 0b110, that selects an OA size of 52 bits, is permitted only when using the 64KB translation granule.
- When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of TCR_EL2.IPS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of TCR_EL2.IPS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

**Note**

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any TTBR1_EL2[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using TTBR1_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of TCR_EL2.T1SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

**CnP, bit [0]**

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1.
<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by TTBR1_EL2 for the current ASID are permitted to differ from corresponding entries for TTBR1_EL2 for other PEs in the Inner Shareable domain. This is not affected by:</td>
</tr>
<tr>
<td></td>
<td>• The value of TTBR1_EL2.CnP on those other PEs.</td>
</tr>
<tr>
<td></td>
<td>• The value of the current ASID.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by TTBR1_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1 and all of the following apply:</td>
</tr>
<tr>
<td></td>
<td>• The translation table entries are pointed to by TTBR1_EL2.</td>
</tr>
<tr>
<td></td>
<td>• The ASID is the same as the current ASID.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

**Note**

- TTBR1_EL2 is accessible only when the value of HCR_EL2.E2H is 1, meaning the current translation regime is the EL2&0 regime.
- If the value of the TTBR1_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Accessing the TTBR1_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic TTBR1_EL2 or TTBR1_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, TTBR1_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return TTBR1_EL2;
elif PSTATE.EL == EL3 then
  return TTBR1_EL2;

**MSR TTBR1_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TTBR1_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  TTBR1_EL2 = X[t];

MRS <Xt>, TTBR1_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TTBR1_EL1 == '1' then
    return NVMem[0x210];
  else
    return TTBR1_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    return TTBR1_EL2;
  else
    return TTBR1_EL1;
elsif PSTATE.EL == EL3 then
  return TTBR1_EL1;

MSR TTBR1_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGWTR_EL2.TTBR1_EL1 == '1' then
    return NVMem[0x210];
  else
    return TTBR1_EL1;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' then
    TTBR1_EL2 = X[t];
  else
    TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  TTBR1_EL1 = X[t];
UAO, User Access Override

The UAO characteristics are:

**Purpose**

Allows access to the User Access Override bit.

**Configuration**

This register is present only when ARMv8.2-UAO is implemented. Otherwise, direct accesses to UAO are **UNDEFINED**.

**Attributes**

UAO is a 64-bit register.

**Field descriptions**

The UAO bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES | RES | UAO | RES | RES |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:24]**

Reserved, RES0.

**UA0, bit [23]**

User Access Override.

<table>
<thead>
<tr>
<th>UAO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The behavior of LDTR* and STTR* instructions is as defined in the base Armv8 architecture.</td>
</tr>
<tr>
<td>0b1</td>
<td>When executed at EL1, or at EL2 with \texttt{HCR_EL2.(E2H, TGE) {1, 1}}, LDTR* and STTR* instructions behave as the equivalent LDR* and STR* instructions.</td>
</tr>
</tbody>
</table>

When executed at EL3, or at EL2 with \texttt{HCR_EL2.E2H} == 0 or PSTATE.UAO bit.

**Bits [22:0]**

Reserved, RES0.

**Accessing the UAO**

For details on the operation of the MSR (immediate) accessor, see MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:
MRS $<Xt>$, UAO

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  return Zeros(40):PSTATE.UAO:Zeros(23);
elsif PSTATE.EL == EL2 then
  return Zeros(40):PSTATE.UAO:Zeros(23);
elsif PSTATE.EL == EL3 then
  return Zeros(40):PSTATE.UAO:Zeros(23);

MSR UAO, $<Xt>$

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  PSTATE.UAO = X[$t$]<23>;
elsif PSTATE.EL == EL2 then
  PSTATE.UAO = X[$t$]<23>;
elsif PSTATE.EL == EL3 then
  PSTATE.UAO = X[$t$]<23>;

MSR UAO, #$<imm>#

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
VBAR_EL1, Vector Base Address Register (EL1)

The VBAR_EL1 characteristics are:

**Purpose**

Holds the vector base address for any exception that is taken to EL1.

**Configuration**

AArch64 System register VBAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register `VBAR[31:0]`. RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

VBAR_EL1 is a 64-bit register.

**Field descriptions**

The VBAR_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector Base Address</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:11]**

Vector Base Address. Base address of the exception vectors for exceptions taken to EL1.

If the implementation does not support ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

If the implementation supports ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

This field resets to an architecturally **UNKNOWN** value.

**Bits [10:0]**

Reserved, RES0.

**Accessing the VBAR_EL1**

When `HCR_EL2.E2H` is 1, without explicit synchronization, access from EL3 using the mnemonic `VBAR_EL1` or `VBAR_EL12` are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, VBAR_EL1

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x250];
    else
        return VBAR_EL1;
    end if;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return VBAR_EL2;
    else
        return VBAR_EL1;
    end if;
elsif PSTATE.EL == EL3 then
    return VBAR_EL1;
else
    VBAR_EL1 = X[t];
end if;

MSR VBAR_EL1, <Xt>

MRS <Xt>, VBAR_EL12
if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x250];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE_EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return VBAR_EL1;
    else
        UNDEFINED;
elsif PSTATE_EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        return VBAR_EL1;
    else
        UNDEFINED;

MSR VBAR_EL12, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x250] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE_EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        VBAR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE_EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        VBAR_EL1 = X[t];
    else
        UNDEFINED;
**VBAR_EL2, Vector Base Address Register (EL2)**

The VBAR_EL2 characteristics are:

**Purpose**

Holds the vector base address for any exception that is taken to EL2.

**Configuration**

AArch64 System register VBAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register HVBAR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

VBAR_EL2 is a 64-bit register.

**Field descriptions**

The VBAR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Vector Base Address | Vector Base Address | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:11]**

Vector Base Address. Base address of the exception vectors for exceptions taken to EL2.

If the implementation does not support ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

If the implementation supports ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.

This field resets to an architecturally **UNKNOWN** value.

**Bits [10:0]**

Reserved, RES0.
Accessing the VBAR_EL2

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic VBAR_EL2 or VBAR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:

**MRS <Xt>, VBAR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VBAR_EL2;
elsif PSTATE.EL == EL3 then
    return VBAR_EL2;

**MSR VBAR_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VBAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VBAR_EL2 = X[t];

**MRS <Xt>, VBAR_EL1**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return NVMem[0x250];
    end if;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return VBAR_EL2;
    else
        return VBAR_EL1;
    end if;
elsif PSTATE.EL == EL3 then
    return VBAR_EL1;
else
    return VBAR_EL1;
endif

MSR VBAR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1> == '01' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGWR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        VBAR_EL1 = X[t];
    end if;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        VBAR_EL2 = X[t];
    else
        VBAR_EL1 = X[t];
    end if;
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t];
else
    VBAR_EL1 = X[t];
endif
The VBAR_EL3 characteristics are:

**Purpose**

Holds the vector base address for any exception that is taken to EL3.

**Configuration**

This register is present only when HaveEL(EL3). Otherwise, direct accesses to VBAR_EL3 are **UNDEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

VBAR_EL3 is a 64-bit register.

**Field descriptions**

The VBAR_EL3 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**Bits [63:11]**

Vector Base Address. Base address of the exception vectors for exceptions taken to EL3.

If the implementation does not support ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

If the implementation supports ARMv8.2-LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL3 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL3 must be the same or else the use of the vector address will result in a recursive exception.

This field resets to an architecturally **UNKNOWN** value.

**Bits [10:0]**

Reserved, **RES0**.

**Accessing the VBAR_EL3**

Accesses to this register use the following encodings:
MRS <Xt>, VBAR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elif PSTATE.EL == EL2 then
  UNDEFINED;
elif PSTATE.EL == EL3 then
  return VBAR_EL3;

MSR VBAR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elif PSTATE.EL == EL2 then
  UNDEFINED;
elif PSTATE.EL == EL3 then
  VBAR_EL3 = X[t];
VDISR_EL2, Virtual Deferred Interrupt Status Register

The VDISR_EL2 characteristics are:

**Purpose**

Records that a virtual SError interrupt has been consumed by an ESB instruction executed at EL1.

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of DISR_EL1 or DISR occurring in program order after the ESB instruction.

**Configuration**

AArch64 System register VDISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDISR[31:0]. This register is present only when RAS is implemented. Otherwise, direct accesses to VDISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is res0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VDISR_EL2 is a 64-bit register.

**Field descriptions**

The VDISR_EL2 bit assignments are:

**When !ELUsingAArch32(EL1):**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-32</td>
<td>RES0</td>
<td>Reserved, res0.</td>
</tr>
<tr>
<td>31-25</td>
<td>IDS</td>
<td>Set to 1 when an ESB instruction defers a virtual SError interrupt. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>24</td>
<td>A</td>
<td>Set to 1 when an ESB instruction defers a virtual SError interrupt.</td>
</tr>
<tr>
<td>23-18</td>
<td>ISS</td>
<td>The value copied from VSESР_EL2.IDS.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally UNKNOWN value.

**ISS, bits [23:0]**

The value copied from VSESR_EL2.ISS.

This field resets to an architecturally UNKNOWN value.

**When ELUsingAAArch32(EL1) and VDISR_EL2.LPAE == 0:**

<table>
<thead>
<tr>
<th>Bits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31:16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15:14</td>
<td>The value copied from VSESR_EL2.AET.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>The value copied from VSESR_EL2.ExT.</td>
</tr>
<tr>
<td>9:0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10110</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.
The FS field is split as follows:

- FS[4] is VDISR_EL2[10].
- FS[3:0] is VDISR_EL2[3:0].

This field resets to an architecturally UNKNOWN value.

**LPAE, bit [9]**

Format.

Set to TTECR.EAE when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Bits [8:4]**

Reserved, RES0.

**FS[3:0], bits [3:0]**

This field is bits[3:0] of FS[4:0].


**When ELUsingAArch32(EL1) and VDISR_EL2.LPAE == 1:**

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| A   | RES0 | AET | RES0 | Ext | RES0 | LPAE | RES0 | STATUS |

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|

**Bits [63:32]**

Reserved, RES0.

**A, bit [31]**

Set to 1 when an ESB instruction defers a virtual SError interrupt.

This field resets to an architecturally UNKNOWN value.

**Bits [30:16]**

Reserved, RES0.

**AET, bits [15:14]**

The value copied from VSESR_EL2.AET.

This field resets to an architecturally UNKNOWN value.

**Bit [13]**

Reserved, RES0.
**ExT, bit [12]**

The value copied from VSESR_EL2.ExT.

This field resets to an architecturally **UNKNOWN** value.

**Bits [11:10]**

Reserved, RES0.

**LPAE, bit [9]**

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [8:6]**

Reserved, RES0.

**STATUS, bits [5:0]**

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b010001</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the VDISR_EL2**

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 or DISR occurring in program order after the ESB instruction.

Accesses to this register use the following encodings:

**MRS <Xt>, VDISR_EL2**

```c
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
      return NVMem[0x500];
   elsif EL2Enabled() && HCR_EL2.NV == '1' then
      AArch64.SystemAccessTrap(EL2, 0x18);
   else
      UNDEFINED;
   endif
elsif PSTATE.EL == EL2 then
   return VDISR_EL2;
elsif PSTATE.EL == EL3 then
   return VDISR_EL2;
```

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
MSR VDISR_EL2, <Xt>

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x500] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  VDISR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  VDISR_EL2 = X[t];

MRS <Xt>, DISR_EL1

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
    return VDISR_EL2;
  else
    return DISR_EL1;
elsif PSTATE.EL == EL2 then
  return DISR_EL1;
elsif PSTATE.EL == EL3 then
  return DISR_EL1;

MSR DISR_EL1, <Xt>

```
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
```

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
    VDISR_EL2 = X[t];
  else
    DISR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
  DISR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  DISR_EL1 = X[t];
VMPIDR_EL2, Virtualization Multiprocessor ID Register

The VMPIDR_EL2 characteristics are:

**Purpose**

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by EL1 reads of MPIDR_EL1.

**Configuration**

AArch64 System register VMPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VMPIDR[31:0].

If EL2 is not implemented, reads of this register return the value of the MPIDR_EL1, and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VMPIDR_EL2 is a 64-bit register.

**Field descriptions**

The VMPIDR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:40]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aff3 is not supported in AArch32 state.</td>
<td></td>
</tr>
<tr>
<td>This field resets to an architecturally UNKNOWN value.</td>
<td></td>
</tr>
<tr>
<td><strong>Bit [31]</strong></td>
<td>Reserved, RES1.</td>
</tr>
<tr>
<td><strong>U, bit [30]</strong></td>
<td>Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system. The possible values of this bit are:</td>
</tr>
<tr>
<td><strong>U</strong></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>0b0</td>
<td>Processor is part of a multiprocessor system.</td>
</tr>
<tr>
<td>0b1</td>
<td>Processor is part of a uniprocessor system.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally `UNKNOWN` value.

**Bits [29:25]**

Reserved, RES0.

**MT, bit [24]**

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels. The possible values of this bit are:

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Performance of PEs at the lowest affinity level is largely independent.</td>
</tr>
<tr>
<td>0b1</td>
<td>Performance of PEs at the lowest affinity level is very interdependent.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally `UNKNOWN` value.

**Aff2, bits [23:16]**

Affinity level 2. See the description of Aff0 for more information.

This field resets to an architecturally `UNKNOWN` value.

**Aff1, bits [15:8]**

Affinity level 1. See the description of Aff0 for more information.

This field resets to an architecturally `UNKNOWN` value.

**Aff0, bits [7:0]**

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

This field resets to an architecturally `UNKNOWN` value.

**Accessing the VMPIDR_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, VMPIDR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x050];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        return VMPIDR_EL2;
    elsif PSTATE.EL == EL3 then
        if !HaveEL(EL2) then
            return MPIDR_EL1;
        else
            return VMPIDR_EL2;
        endif
    endif
else
    UNDEFINED;
endif

MSR VMPIDR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x050] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        VMPIDR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        if !HaveEL(EL2) then
            // no operation
        else
            VMPIDR_EL2 = X[t];
        endif
    endif
else
    UNDEFINED;
endif

MRS <Xt>, MPIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        end if;
    else
        UNDEFINED;
    end if;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
        HFGRTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VMPIDR_EL2;
    else
        return MPIDR_EL1;
    end if;
elsif PSTATE.EL == EL2 then
    return MPIDR_EL1;
else
    return MPIDR_EL1;
end if;

MRC{<c>}{<q>} {<coproc>}, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VMPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VMPIDR<31:0>;
    else
        return MPIDR<31:0>;
    end if;
elsif PSTATE.EL == EL2 then
    return MPIDR<31:0>;
else
    return MPIDR<31:0>;
end if;
VNCR_EL2, Virtual Nested Control Register

The VNCR_EL2 characteristics are:

**Purpose**

When ARMv8.4-NV is implemented, holds the base address that is used to define the memory location that is accessed by transformed reads and writes of System registers.

**Configuration**

This register is present only when ARMv8.4-NV is implemented. Otherwise, direct accesses to VNCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VNCR_EL2 is a 64-bit register.

**Field descriptions**

The VNCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit (63:53)</th>
<th>Bit (52:12)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESS</td>
<td>BADDR</td>
</tr>
<tr>
<td></td>
<td>RES0</td>
</tr>
</tbody>
</table>

**RESS, bits [63:53]**

Reserved, Sign extended. If the bits marked as RESS do not all have the same value, then there is a CONSTRAINED UNPREDICTABLE choice between:

- Generating an EL2 translation regime Translation abort on use of the VNCR_EL2 register.
- Bits[63:49] of VNCR_EL2 are treated as the same value as bit[48] for all purposes other than reading back the register.
- Bits[63:49] of VNCR_EL2 are treated as the same value as bit[48] for all purposes.
- If the virtual address space for EL2 supports more than 48 bits, bits[63:53] of VNCR_EL2 are treated as the same value as bit[52] for all purposes other than reading back the register.
- If the virtual address space for EL2 supports more than 48 bits, bits[63:53] of VNCR_EL2 are treated as the same value as bit[52].

Where the EL2 translation regime has upper and lower address ranges, bit[52] is used to select between those address ranges to determine if the address space supports more than 48 bits.

**BADDR, bits [52:12]**

Base Address. If the virtual address space for EL2 does not support more than 48 bits, then bits [52:49] are RESS.

When a register read/write is transformed to be a Load or Store, the address of the load/store is to SignOffset(VNCR.BADDR.Offset<11:0>, 64).

This field resets to an architecturally UNKNOWN value.
Bits [11:0]

Reserved, RES0.

Accessing the VNCR_EL2

Accesses to this register use the following encodings:

**MRS <Xt>, VNCR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x0B0];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else
  if PSTATE.EL == EL2 then
    return VNCR_EL2;
  elsif PSTATE.EL == EL3 then
    return VNCR_EL2;

**MSR VNCR_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x0B0] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
else
  if PSTATE.EL == EL2 then
    VNCR_EL2 = X[t];
  elsif PSTATE.EL == EL3 then
    VNCR_EL2 = X[t];
VPIDR_EL2, Virtualization Processor ID Register

The VPIDR_EL2 characteristics are:

**Purpose**

Holds the value of the Virtualization Processor ID. This is the value returned by EL1 reads of MIDR_EL1.

**Configuration**

AArch64 System register VPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VPIDR[31:0].

If EL2 is not implemented, reads of this register return the value of the MIDR_EL1, and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

**Attributes**

VPIDR_EL2 is a 64-bit register.

**Field descriptions**

The VPIDR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:32]</td>
<td>RESERVED, RES0</td>
</tr>
<tr>
<td>[31:24]</td>
<td>Implementer</td>
</tr>
<tr>
<td>[23:16]</td>
<td>Variant</td>
</tr>
<tr>
<td>[15:8]</td>
<td>Architecture</td>
</tr>
<tr>
<td>[7:0]</td>
<td>PartNum</td>
</tr>
</tbody>
</table>

**Implementer, bits [31:24]**

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

<table>
<thead>
<tr>
<th>Hex representation</th>
<th>ASCII representation</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x41</td>
<td>A</td>
<td>Arm Limited</td>
</tr>
<tr>
<td>0x42</td>
<td>B</td>
<td>Broadcom Corporation</td>
</tr>
<tr>
<td>0x43</td>
<td>C</td>
<td>Cavium Inc.</td>
</tr>
<tr>
<td>0x44</td>
<td>D</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>0x49</td>
<td>I</td>
<td>Infineon Technologies AG</td>
</tr>
<tr>
<td>0x4D</td>
<td>M</td>
<td>Motorola or Freescale</td>
</tr>
<tr>
<td>0x50</td>
<td>P</td>
<td>Applied Micro Circuits Corporation</td>
</tr>
<tr>
<td>0x51</td>
<td>Q</td>
<td>Qualcomm Inc.</td>
</tr>
<tr>
<td>0x56</td>
<td>V</td>
<td>Marvell International Ltd.</td>
</tr>
<tr>
<td>0x69</td>
<td>i</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.
This field resets to an architecturally **UNKNOWN** value.

**Variant, bits [23:20]**

An **IMPLEMENTATION DEFINED** variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field resets to an architecturally **UNKNOWN** value.

**Architecture, bits [19:16]**

The permitted values of this field are:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Armv4.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Armv4T.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Armv5 (obsolete).</td>
</tr>
<tr>
<td>0b0100</td>
<td>Armv5T.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Armv5TE.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Armv5TEJ.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv6.</td>
</tr>
</tbody>
</table>
| 0b1111       | Architectural features are individually identified in the | | ID * registers, see 'ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K12.3.3. All other values are reserved. This field resets to an architecturally **UNKNOWN** value.

**PartNum, bits [15:4]**

An **IMPLEMENTATION DEFINED** primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field resets to an architecturally **UNKNOWN** value.

**Revision, bits [3:0]**

An **IMPLEMENTATION DEFINED** revision number for the device.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the VPIDR_EL2**

Accesses to this register use the following encodings:

\[
\text{MRS <Xt>, VPIDR_EL2}
\]

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x088];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end if;
elsif PSTATE.EL == EL2 then
    return VPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return MIDR_EL1;
    else
        return VPIDR_EL2;
    end if;
end if;

MSR VPIDR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x088] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    end if;
elsif PSTATE.EL == EL2 then
    VPIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        //no operation
    else
        VPIDR_EL2 = X[t];
    end if;

MRS <Xt>, MIDR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    if IsFeatureImplemented("ARMv8.4-IDST") then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    end
    else
        UNDEFINED;
    end
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
    HFGRTR_EL2.MIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2;
    else
        return MIDR_EL1;
    end
elsif PSTATE_EL == EL2 then
    return MIDR_EL1;
elsif PSTATE_EL == EL3 then
    return MIDR_EL1;
endif

MRC{<c>}{<q>}{<coproc>}, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VPIDR<31:0>;
    else
        return MIDR<31:0>;
    end
elsif PSTATE_EL == EL2 then
    return MIDR<31:0>;
elsif PSTATE_EL == EL3 then
    return MIDR<31:0>;
endif

VPIDR_EL2, Virtualization Processor ID Register
The VSESR_EL2 characteristics are:

**Purpose**

Provides the syndrome value reported to software on taking a virtual SError interrupt exception to EL1, or on executing an ESB instruction at EL1.

When the virtual SError interrupt is taken to EL1 using AArch64, then the syndrome value is reported in ESR_EL1.

When the virtual SError interrupt is taken to EL1 using AArch32, then the syndrome value is reported in DFSR. {AET, ExT} and the remainder of DFSR is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

When the virtual SError interrupt is deferred by an ESB instruction, then the syndrome value is written to VDISR_EL2.

**Configuration**

AArch64 System register VSESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VDFSR[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to VSESR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VSESR_EL2 is a 64-bit register.

**Field descriptions**

The VSESR_EL2 bit assignments are:

**When ELUsingAArch32(EL1):**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Bits [63:16]

Reserved, RES0.

**AET, bits [15:14]**

When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[15:4] is set to VSESR_EL2.AET.

When a virtual SError interrupt is deferred by an ESB instruction, VDISR_EL2[15:4] is set to VSESR_EL2.AET.

This field resets to an architecturally UNKNOWN value.
Bit [13]
Reserved, RES0.

ExT, bit [12]
When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[12] is set to VSESR_EL2.ExT.
When a virtual SError interrupt is deferred by an ESB instruction, VDIRS_EL2[12] is set to VSESR_EL2.ExT.
This field resets to an architecturally UNKNOWN value.

Bits [11:0]
Reserved, RES0.

**When !ELUsingAArch32(EL1):**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>IDS</td>
</tr>
<tr>
<td>61</td>
<td>ISS</td>
</tr>
<tr>
<td>60</td>
<td>......</td>
</tr>
<tr>
<td>59</td>
<td>......</td>
</tr>
<tr>
<td>58</td>
<td>......</td>
</tr>
<tr>
<td>57</td>
<td>......</td>
</tr>
<tr>
<td>56</td>
<td>......</td>
</tr>
<tr>
<td>55</td>
<td>......</td>
</tr>
<tr>
<td>54</td>
<td>......</td>
</tr>
<tr>
<td>53</td>
<td>......</td>
</tr>
<tr>
<td>52</td>
<td>......</td>
</tr>
<tr>
<td>51</td>
<td>......</td>
</tr>
<tr>
<td>50</td>
<td>......</td>
</tr>
<tr>
<td>49</td>
<td>......</td>
</tr>
<tr>
<td>48</td>
<td>......</td>
</tr>
<tr>
<td>47</td>
<td>......</td>
</tr>
<tr>
<td>46</td>
<td>......</td>
</tr>
<tr>
<td>45</td>
<td>......</td>
</tr>
<tr>
<td>44</td>
<td>......</td>
</tr>
<tr>
<td>43</td>
<td>......</td>
</tr>
<tr>
<td>42</td>
<td>......</td>
</tr>
<tr>
<td>41</td>
<td>......</td>
</tr>
<tr>
<td>40</td>
<td>......</td>
</tr>
<tr>
<td>39</td>
<td>......</td>
</tr>
<tr>
<td>38</td>
<td>......</td>
</tr>
<tr>
<td>37</td>
<td>......</td>
</tr>
<tr>
<td>36</td>
<td>......</td>
</tr>
<tr>
<td>35</td>
<td>......</td>
</tr>
<tr>
<td>34</td>
<td>......</td>
</tr>
<tr>
<td>33</td>
<td>......</td>
</tr>
<tr>
<td>32</td>
<td>......</td>
</tr>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>......</td>
</tr>
<tr>
<td>29</td>
<td>......</td>
</tr>
<tr>
<td>28</td>
<td>......</td>
</tr>
<tr>
<td>27</td>
<td>......</td>
</tr>
<tr>
<td>26</td>
<td>......</td>
</tr>
<tr>
<td>25</td>
<td>......</td>
</tr>
<tr>
<td>24</td>
<td>......</td>
</tr>
<tr>
<td>23</td>
<td>......</td>
</tr>
<tr>
<td>22</td>
<td>......</td>
</tr>
<tr>
<td>21</td>
<td>......</td>
</tr>
<tr>
<td>20</td>
<td>......</td>
</tr>
<tr>
<td>19</td>
<td>......</td>
</tr>
<tr>
<td>18</td>
<td>......</td>
</tr>
<tr>
<td>17</td>
<td>......</td>
</tr>
<tr>
<td>16</td>
<td>......</td>
</tr>
<tr>
<td>15</td>
<td>......</td>
</tr>
<tr>
<td>14</td>
<td>......</td>
</tr>
<tr>
<td>13</td>
<td>......</td>
</tr>
<tr>
<td>12</td>
<td>......</td>
</tr>
<tr>
<td>11</td>
<td>......</td>
</tr>
<tr>
<td>10</td>
<td>......</td>
</tr>
<tr>
<td>9</td>
<td>......</td>
</tr>
<tr>
<td>8</td>
<td>......</td>
</tr>
<tr>
<td>7</td>
<td>......</td>
</tr>
<tr>
<td>6</td>
<td>......</td>
</tr>
<tr>
<td>5</td>
<td>......</td>
</tr>
<tr>
<td>4</td>
<td>......</td>
</tr>
<tr>
<td>3</td>
<td>......</td>
</tr>
<tr>
<td>2</td>
<td>......</td>
</tr>
<tr>
<td>1</td>
<td>......</td>
</tr>
<tr>
<td>0</td>
<td>......</td>
</tr>
</tbody>
</table>

Bits [63:25]
Reserved, RES0.

IDS, bit [24]
When a virtual SError interrupt is taken to EL1 using AArch64, ESR_EL1[24] is set to VSESR_EL2.IDS.
When a virtual SError interrupt is deferred by an ESB instruction, VDIRS_EL2[24] is set to VSESR_EL2.IDS.
This field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]
When a virtual SError interrupt is taken to EL1 using AArch64, ESR_EL1[23:0] is set to VSESR_EL2.ISS.
When a virtual SError interrupt is deferred by an ESB instruction, VDIRS_EL2[23:0] is set to VSESR_EL2.ISS.
This field resets to an architecturally UNKNOWN value.

**Accessing the VSESR_EL2**

Accesses to this register use the following encodings:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x508];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    return VSESR_EL2;
elsif PSTATE.EL == EL3 then
    return VSESR_EL2;
endif

MSR VSESR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x508] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    VSESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VSESR_EL2 = X[t];
VSTCR_EL2, Virtualization Secure Translation Control Register

The VSTCR_EL2 characteristics are:

**Purpose**

The control register for stage 2 of the Secure EL1&0 translation regime.

**Configuration**

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to VSTCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VSTCR_EL2 is a 64-bit register.

**Field descriptions**

The VSTCR_EL2 bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 | Reserved, RES0 |

Any of the bits in VSTCR_EL2 are permitted to be cached in a TLB.

**Bits [63:32]**

Reserved, RES0.

**Bit [31]**

Reserved, RES1.

**SA, bit [30]**

Secure stage 2 translation output address space.

<table>
<thead>
<tr>
<th>SA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All stage 2 translations for the Secure IPA space access the Secure PA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>All stage 2 translations for the Secure IPA space access the Non-secure PA space.</td>
</tr>
</tbody>
</table>

When the value of VSTCR_EL2.SW is 1, this bit behaves as 1 for all purposes other than reading back the value of the bit.

This field resets to an architecturally UNKNOWN value.
SW, bit [29]

Secure stage 2 translation address space.

<table>
<thead>
<tr>
<th>SW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All stage 2 translation table walks for the Secure IPA space are to the Secure PA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>All stage 2 translation table walks for the Secure IPA space are to the Non-secure PA space.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [28:16]**

Reserved, RES0.

**TG0, bits [15:14]**

Secure stage 2 granule size for **VSTTBR_EL2**.

<table>
<thead>
<tr>
<th>TG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b01</td>
<td>64KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>16KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

If ARMv8.5-GTG is implemented, **ID_AA64MMFR0_EL1**.{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for Level 2 translation.

If ARMv8.5-GTG is not implemented, **ID_AA64MMFR0_EL1**.{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value, or a size that has not been implemented, then for all purposes other than read back from this register, the hardware will treat the field as if it has been programmed to an **IMPLEMENTATION DEFINED** choice of the sizes that has been implemented.

It is **IMPLEMENTATION DEFINED** whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally **UNKNOWN** value.

**Bits [13:8]**

Reserved, RES0.

**SL0, bits [7:6]**

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

<table>
<thead>
<tr>
<th>SL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.</td>
</tr>
<tr>
<td>0b01</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.</td>
</tr>
<tr>
<td>0b10</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.</td>
</tr>
<tr>
<td>0b11</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 3.</td>
</tr>
</tbody>
</table>
All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

<table>
<thead>
<tr>
<th>SL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3.</td>
</tr>
<tr>
<td>0b01</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2.</td>
</tr>
<tr>
<td>0b10</td>
<td>If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1.</td>
</tr>
</tbody>
</table>

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

T0SZ, bits [5:0]

The size offset of the memory region addressed by VSTTBR_EL2. The region size is $2^{(64-VSTCR_EL2.T0SZ)}$ bytes.

The maximum and minimum possible values for this field depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

**Accessing the VSTCR_EL2**

Accesses to this register use the following encodings:

MRS <Xt>, VSTCR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x048];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    return VSTCR_EL2;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    return VSTCR_EL2;

MSR VSTCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x048] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    VSTCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    VSTCR_EL2 = X[t];
VSTTBR_EL2, Virtualization Secure Translation Table Base Register

The VSTTBR_EL2 characteristics are:

**Purpose**

The base register for stage 2 of the Secure EL1&0 translation regime. Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the Secure EL1&0 translation regime, and other information for this translation stage.

**Configuration**

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to VSTTBR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VSTTBR_EL2 is a 64-bit register.

**Field descriptions**

The VSTTBR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>BADDR</td>
</tr>
<tr>
<td>61</td>
<td>BADDR</td>
</tr>
<tr>
<td>60</td>
<td>CnP</td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Any of the bits in VSTTBR_EL2 are permitted to be cached in a TLB.

**Bits [63:48]**

Reserved, RES0.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x].

**Note**

- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

If the value of VTCR_EL2.PS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
When z>x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
When x>6 register bits[(x-1):6] are RES0.
Register bit[1] is RES0.
Bits[5:2] of the stage 1 translation table base address are zero.

Note
When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of VTCR_EL2.PS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of VTCR_EL2.PS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

If any VSTTBR_EL2[47:1] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VSTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VSTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes ARMv8.2-TTNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.</td>
</tr>
</tbody>
</table>

Note
If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values on page K1-6254.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

This field resets to an architecturally UNKNOWN value.
Accessing the **VSTTBR_EL2**

Accesses to this register use the following encodings:

**MRS <Xt>, VSTTBR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  elsif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x030];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && SCR_EL3.NS == '1' then
    UNDEFINED;
  else
    return VSTTBR_EL2;
elsif PSTATE.EL == EL3 then
  if SCR_EL3.EEL2 == '0' then
    UNDEFINED;
  else
    return VSTTBR_EL2;
```
VTCR_EL2, Virtualization Translation Control Register

The VTCR_EL2 characteristics are:

**Purpose**

The control register for stage 2 of the EL1&0 translation regime.

**Configuration**

AArch64 System register VTCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register VTCR[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VTCR_EL2 is a 64-bit register.

**Field descriptions**

The VTCR_EL2 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | NSA | ANSW | HWU62 | HWU61 | HWU60 | HWU59 | RES0 | HD | HA | RES0 | VSS | PS | TG0 | SH0 | ORGN0 | RGN0 | RG0 | SL0 | T0S | Z |

Any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

**Bits [63:32]**

Reserved, RES0.

**Bit [31]**

Reserved, RES1.

**NSA, bit [30]**

When ARMv8.4-SecEL2 is implemented:

Non-secure stage 2 translation output address space.

<table>
<thead>
<tr>
<th>NSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All stage 2 translations for the Non-secure IPA space of the Secure EL1&amp;0 translation regime access the Secure PA space.</td>
</tr>
<tr>
<td>0b1</td>
<td>All stage 2 translations for the Non-secure IPA space of the Secure EL1&amp;0 translation regime access the Non-secure PA space.</td>
</tr>
</tbody>
</table>

This bit behaves as 1 for all purposes other than reading back the value of the bit when one of the following is true:

- The PE is executing in Non-secure state.
- The value of VTCR_EL2.NSW is 1.
- The value of VSTCR_EL2.SA is 1.
This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**NSW, bit [29]**

*When ARMv8.4-SecEL2 is implemented:*

Non-secure stage 2 translation table address space.

<table>
<thead>
<tr>
<th>NSW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&amp;0 translation regime are to the Secure PA space.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&amp;0 translation regime are to the Non-secure PA space.</td>
</tr>
</tbody>
</table>

When the PE is executing in Non-secure state, this bit behaves as 1 for all purposes other than reading back the value of the bit.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HWU62, bit [28]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates **IMPLEMENTATION DEFINED** hardware use of bit[62] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU62</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an <strong>IMPLEMENTATION DEFINED</strong> purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HWU61, bit [27]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates **IMPLEMENTATION DEFINED** hardware use of bit[61] of the stage 2 translation table Block or Page entry.
<table>
<thead>
<tr>
<th>HWU61</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**HWU60, bit [26]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU60</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**HWU59, bit [25]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU59</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [24:23]**

Reserved, RES0.

**HD, bit [22]**
When ARMv8.1-TTHM is implemented:

Hardware management of dirty state in stage 2 translations when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>HD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 2 hardware management of dirty state disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 2 hardware management of dirty state enabled, only if the VTCR_EL2.HA bit is also set to 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**HA, bit [21]**

When ARMv8.1-TTHM is implemented:

Hardware Access flag update in Non-secure and Secure stage 2 translations when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>HA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stage 2 Access flag update disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stage 2 Access flag update enabled.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**Bit [20]**

Reserved, RES0.

**VS, bit [19]**

When ARMv8.1-VMID16 is implemented:

VMID Size.

<table>
<thead>
<tr>
<th>VS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>8 bit - the upper 8 bits of VTTBR_EL2 and VSTTBR_EL2 are ignored by the hardware, and treated as if they are all zeros, for every purpose except when reading back the register.</td>
</tr>
<tr>
<td>0b1</td>
<td>16 bit - the upper 8 bits of VTTBR_EL2 and VSTTBR_EL2 are used for allocation and matching in the TLB.</td>
</tr>
</tbody>
</table>

If the implementation only supports an 8-bit VMID, this field is RES0.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PS, bits [18:16]**

Physical address Size for the Second Stage of translation.
### PS, Meaning

<table>
<thead>
<tr>
<th>PS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>32 bits, 4GB.</td>
</tr>
<tr>
<td>0b01</td>
<td>36 bits, 64GB.</td>
</tr>
<tr>
<td>0b10</td>
<td>40 bits, 1TB.</td>
</tr>
<tr>
<td>0b11</td>
<td>42 bits, 4TB.</td>
</tr>
<tr>
<td>0b100</td>
<td>44 bits, 16TB.</td>
</tr>
<tr>
<td>0b101</td>
<td>48 bits, 256TB.</td>
</tr>
<tr>
<td>0b110</td>
<td>52 bits, 4PB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

The value 0b110 is permitted only if ARMv8.2-LPA is implemented and the translation granule size is 64KB.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by VTCR_EL2 are 0b0000.

This field resets to an architecturally UNKNOWN value.

### TG0, bits [15:14]

Granule size for the VTTBR_EL2.

<table>
<thead>
<tr>
<th>TG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b01</td>
<td>64KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>16KB.</td>
</tr>
</tbody>
</table>

Other values are reserved.

If ARMv8.5-GTG is implemented, ID_AA64MMFR0_EL1.{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for Level 2 translation.

If ARMv8.5-GTG is not implemented, ID_AA64MMFR0_EL1.{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

This field resets to an architecturally UNKNOWN value.

### SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

This field resets to an architecturally UNKNOWN value.

### ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in AArch64 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

This field resets to an architecturally UNKNOWN value.
ORGN0 | Meaning
--- | ---
0b00 | Normal memory, Outer Non-cacheable.
0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.
0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.
0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.

This field resets to an architecturally UNKNOWN value.

**IRGN0, bits [9:8]**

Inner cacheability attribute for memory associated with translation table walks using VTTBR_EL2 or VSTTBR_EL2.

| IRGN0 | Meaning |
--- | ---
0b00 | Normal memory, Inner Non-cacheable. |
0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

This field resets to an architecturally UNKNOWN value.

**SL0, bits [7:6]**

When ARMv8.4-TTST is implemented:

Starting level of the Secure stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

| SL0 | Meaning |
--- | ---
0b00 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3. |
0b01 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2. |
0b10 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1. |
0b11 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 3. |

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Starting level of the Secure stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

| SL0 | Meaning |
--- | ---
0b00 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3. |
0b01 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2. |
0b10 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1. |
All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally **UNKNOWN** value.

**T0SZ, bits [5:0]**

The size offset of the memory region addressed by VTTBR_EL2. The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0 then a stage 2 level 0 Translation fault is generated.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the VTCR_EL2**

Any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings:

**MRS <Xt>, VTCR_EL2**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x040];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTCR_EL2;
elsif PSTATE.EL == EL3 then
    return VTCR_EL2;

**MSR VTCR_EL2, <Xt>**

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x040] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VTCR_EL2 = X[t];
VTTBR_EL2, Virtualization Translation Table Base Register

The VTTBR_EL2 characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the EL1&0 translation regime, and other information for this translation regime.

**Configuration**

AArch64 System register VTTBR_EL2 bits [63:0] are architecturally mapped to AArch32 System register VTTBR[63:0].

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VTTBR_EL2 is a 64-bit register.

**Field descriptions**

The VTTBR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>VMID[15:8]</td>
<td>VMID[7:0]</td>
<td>BADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**VMID[15:8], bits [63:56]**

When ARMv8.1-VMID16 is implemented:

Extension to VMID[7:0]. See VMID[7:0] for more details.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**VMID[7:0], bits [55:48]**

The VMID for the translation table.

It is IMPLEMENTATION DEFINED whether the VMID is 8 bits or 16 bits.

If the implementation has an 8-bit VMID, then VMID[15:8] are RES0.

If the implementation has a 16-bit VMID, then:
The VTCR_EL2 VS bit selects whether VMID[15:8] are ignored by the hardware for every purpose except reading back the register, or whether these bits are used for allocation and matching in the TLB.

The 16-bit VMID is only supported when EL2 is using AArch64. This means the hardware must ignore VMID[15:8] when EL2 is using AArch32.

This field resets to an architecturally UNKNOWN value.

**BADDR, bits [47:1]**

Translation table base address, A[47:x] or A[51:x], bits[47:1].

**Note**
- Translation table base addresses of 52 bits, A[51:x], are supported only in an implementation that includes ARMv8.2-LPA and is using the 64KB translation granule.
- A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes ARMv8.2-LPA, if the value of VTCR_EL2.PS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
  - If x >= 6 then z=x.
  - Otherwise, z=6.
- When z=x register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When x>6 register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.
- In an implementation that includes ARMv8.2-TTCNP, bit[0] of the stage 1 translation table base address is zero.

**Note**
- In an implementation that includes ARMv8.2-LPA a VTCR_EL2 PS value of 0b110, that selects a PA size of 52 bits, is permitted only when using the 64KB translation granule.
- When the value of ID_AA64MMFR0_EL1.PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the value of VTCR_EL2.PS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of VTCR_EL2.PS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

**Note**
This definition applies:

- To an implementation that includes ARMv8.2-LPA and is using a translation granule smaller than 64KB.
- To any implementation that does not include ARMv8.2-LPA.

If any VTTBR_EL2[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.
The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of VTCR_EL2.T0SZ, the stage of translation, and the translation granule size.

This field resets to an architecturally **UNKNOWN** value.

### CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by VTTBR_EL2 are permitted to differ from the entries for VTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by VTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

**Note**

If the value of VTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBR_EL2s do not point to the same translation table entries when using the current VMID then the results of translations using VTTBR_EL2 are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

### Accessing the VTTBR_EL2

Accesses to this register use the following encodings:

MRS <Xt>, VTTBR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    return NVMem[0x020];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    return AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  endif
elsif PSTATE.EL == EL2 then
  return VTTBR_EL2;
elsif PSTATE.EL == EL3 then
  return VTTBR_EL2;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
    NVMem[0x020] = X[t];
  elsif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  VTTBR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
  VTTBR_EL2 = X[t];
ZCR_EL1, SVE Control Register for EL1

The ZCR_EL1 characteristics are:

Purpose

The SVE Control Register for EL1 is used to control aspects of SVE visible at Exception levels EL1 and EL0.

Configuration

This register is present only when SVE is implemented. Otherwise, direct accesses to ZCR_EL1 are UNDEFINED.

When HCR_EL2.{E2H, TGE} == {1, 1} and EL2 is enabled in the current Security state, the fields in this register have no effect on execution at EL0

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ZCR_EL1 is a 64-bit register.

Field descriptions

The ZCR_EL1 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | RAZ/WI | LEN |

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Constrains the scalable vector register length for EL1 and EL0 to (LEN+1)x128 bits. For all purposes other than returning the result of a direct read of ZCR_EL1 then this field behaves as if it is set to the minimum of the stored value and the constrained length inherited from more privileged Exception levels in the current Security state, rounded down to the nearest implemented vector length.

An indirect read of ZCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

This field resets to an architecturally UNKNOWN value.

Accessing the ZCR_EL1

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL3 using the mnemonic ZCR_EL1 or ZCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, ZCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL1, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x1E0];
  else
    return ZCR_EL1;
  fi
else
  if PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
      AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' then
      return ZCR_EL2;
    else
      return ZCR_EL1;
    fi
  else
    if PSTATE.EL == EL3 then
      if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
      else
        return ZCR_EL1;
      fi
    else
      return ZCR_EL1;
    fi
  fi
end if

MSR ZCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL1, 0x19);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x1E0] = X[t];
  else
    ZCR_EL1 = X[t];
else PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '1' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elseif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elseif HCR_EL2.E2H == '1' then
    ZCR_EL2 = X[t];
  else
    ZCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    ZCR_EL1 = X[t];
endif

MRS <Xt>, ZCR_EL12

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
    return NVMem[0x1E0];
  elseif EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
      return ZCR_EL1;
    elseif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
      AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
      AArch64.SystemAccessTrap(EL3, 0x19);
    else
      return ZCR_EL1;
    else
      UNDEFINED;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && HCR_EL2.E2H == '1' then
    if CPTR_EL3.EZ == '0' then
      AArch64.SystemAccessTrap(EL3, 0x19);
    else
      return ZCR_EL1;
  else
    UNDEFINED;
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x1E0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && HCR_EL2.E2H == '1' then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t];
    else
        UNDEFINED;

ZCR_EL1, SVE Control Register for EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b101</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

27/09/2019 18:23, 6134483bd14dc8c12a99c984cb6e3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ZCR_EL2, SVE Control Register for EL2

The ZCR_EL2 characteristics are:

**Purpose**

The SVE Control Register for EL2 is used to control aspects of SVE visible at Exception levels EL2, EL1, and EL0, when EL2 is enabled in the current Security state.

**Configuration**

This register is present only when SVE is implemented. Otherwise, direct accesses to ZCR_EL2 are **UNDEFINED**.

This register has no effect if EL2 is not enabled in the current Security state.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ZCR_EL2 is a 64-bit register.

**Field descriptions**

The ZCR_EL2 bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
</table>
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | RES0

**Bits [63:9]**

Reserved, RES0.

**Bits [8:4]**

Reserved, RAZ/WI.

**LEN, bits [3:0]**

Constrains the scalable vector register length for EL2, EL1, and EL0 to (LEN+1)x128 bits, when EL2 is enabled in the current Security state. For all purposes other than returning the result of a direct read of ZCR_EL2 then this field behaves as if it is set to the minimum of the stored value and the constrained length inherited from more privileged Exception levels in the current Security state, rounded down to the nearest implemented vector length.

An indirect read of ZCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ZCR_EL2**

When HCR_EL2.E2H is 1, without explicit synchronization, access from EL2 using the mnemonic ZCR_EL2 or ZCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings:
MRS <Xt>, ZCR_EL2

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

If PSTATE.EL == EL0 then
UNDEFINED;

elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    return ZCR_EL2;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    ZCR_EL2 = X[t];
  endif
endif

MSR ZCR_EL2, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;

elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
  else
    UNDEFINED;
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    ZCR_EL2 = X[t];
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    ZCR_EL2 = X[t];
  endif
endif

MRS <Xt>, ZCR_EL1

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL1, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    return NVMem[0x1E0];
  else
    return ZCR_EL1;
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  elsif HCR_EL2.E2H == '1' then
    return ZCR_EL2;
  else
    return ZCR_EL1;
  endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    return ZCR_EL1;
  endif
msr ZCR_EL1, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL1, 0x19);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<NV2,NV1,NV> == '111' then
    NVMem[0x1E0] = X[t];
  else
    ZCR_EL1 = X[t];
endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
    AArch64.SystemAccessTrap(EL2, 0x19);
  elsif HCR_EL2.E2H == '1' then
    ZCR_EL2 = X[t];
  else
    ZCR_EL1 = X[t];
endif
elsif PSTATE.EL == EL3 then
  if CPTR_EL3.EZ == '0' then
    AArch64.SystemAccessTrap(EL3, 0x19);
  else
    ZCR_EL1 = X[t];
endif
ZCR_EL3, SVE Control Register for EL3

The ZCR_EL3 characteristics are:

Purpose

The SVE Control Register for EL3 is used to control aspects of SVE visible at all Exception levels.

Configuration

This register is present only when SVE is implemented. Otherwise, direct accesses to ZCR_EL3 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

ZCR_EL3 is a 64-bit register.

Field descriptions

The ZCR_EL3 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | RAZ/WI | LEN |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Constrains the scalable vector register length for all Exception levels to (LEN+1)x128 bits. For all purposes other than returning the result of a direct read of ZCR_EL3 then this field behaves as if rounded down to the nearest implemented vector length.

An indirect read of ZCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

This field resets to an architecturally UNKNOWN value.

Accessing the ZCR_EL3

Accesses to this register use the following encodings:

MRS <Xt>, ZCR_EL3

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL3;
    end
end
```

```c
MSR ZCR_EL3, <Xt>

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
```

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL3 = X[t];
    end
end
```
AArch32 System Registers

**ACTLR**: Auxiliary Control Register

**ACTLR2**: Auxiliary Control Register 2

**ADFSR**: Auxiliary Data Fault Status Register

**AIDR**: Auxiliary ID Register

**AIFSR**: Auxiliary Instruction Fault Status Register

**AMAIR0**: Auxiliary Memory Attribute Indirection Register 0

**AMAIR1**: Auxiliary Memory Attribute Indirection Register 1

**AMCFG**: Activity Monitors Configuration Register

**AMCGCR**: Activity Monitors Counter Group Configuration Register

**AMCNTENCLR0**: Activity Monitors Count Enable Clear Register 0

**AMCNTENCLR1**: Activity Monitors Count Enable Clear Register 1

**AMCNTENSET0**: Activity Monitors Count Enable Set Register 0

**AMCNTENSET1**: Activity Monitors Count Enable Set Register 1

**AMCR**: Activity Monitors Control Register

**AMEVCNTR0<\text{n}>**: Activity Monitors Event Counter Registers 0

**AMEVCNTR1<\text{n}>**: Activity Monitors Event Counter Registers 1

**AMEVTYPER0<\text{n}>**: Activity Monitors Event Type Registers 0

**AMEVTYPER1<\text{n}>**: Activity Monitors Event Type Registers 1

**AMUSERENR**: Activity Monitors User Enable Register

**APSR**: Application Program Status Register

**CCSIDR**: Current Cache Size ID Register

**CCSIDR2**: Current Cache Size ID Register 2

**CLIDR**: Cache Level ID Register

**CNTFRQ**: Counter-timer Frequency register

**CNTHCTL**: Counter-timer Hyp Control register

**CNTHPS_CTL**: Counter-timer Secure Physical Timer Control Register (EL2)

**CNTHPS_CVAL**: Counter-timer Secure Physical Timer CompareValue Register (EL2)

**CNTHPS_TVAL**: Counter-timer Secure Physical Timer TimerValue Register (EL2)

**CNTHP_CTL**: Counter-timer Hyp Physical Timer Control register

**CNTHP_CVAL**: Counter-timer Hyp Physical CompareValue register

**CNTHP_TVAL**: Counter-timer Hyp Physical Timer TimerValue register

**CNTHVS_CTL**: Counter-timer Secure Virtual Timer Control Register (EL2)

**CNTHVS_CVAL**: Counter-timer Secure Virtual Timer CompareValue Register (EL2)

**CNTHVS_TVAL**: Counter-timer Secure Virtual Timer TimerValue Register (EL2)
**CNTHV_CTL**: Counter-timer Virtual Timer Control register (EL2)
**CNTHV_CVAL**: Counter-timer Virtual Timer CompareValue register (EL2)
**CNTHV_TVAL**: Counter-timer Virtual Timer TimerValue register (EL2)
**CNTKCTL**: Counter-timer Kernel Control register
**CNTPCT**: Counter-timer Physical Count register
**CNTPCTSS**: Counter-timer Self-Synchronized Physical Count register
**CNTP_CTL**: Counter-timer Physical Timer Control register
**CNTP_CVAL**: Counter-timer Physical Timer CompareValue register
**CNTP_TVAL**: Counter-timer Physical Timer TimerValue register
**CNTVCT**: Counter-timer Virtual Count register
**CNTVCTSS**: Counter-timer Self-Synchronized Virtual Count register
**CNTVOFF**: Counter-timer Virtual Offset register
**CNTV_CTL**: Counter-timer Virtual Timer Control register
**CONTEXTIDR**: Context ID Register
**CPACR**: Architectural Feature Access Control Register
**CPSR**: Current Program Status Register
**CSSELR**: Cache Size Selection Register
**CTR**: Cache Type Register
**DACR**: Domain Access Control Register
**DBGAUTHSTATUS**: Debug Authentication Status register
**DBGBCR<n>**: Debug Breakpoint Control Registers
**DBGVR<n>**: Debug Breakpoint Value Registers
**DBGXVR<n>**: Debug Breakpoint Extended Value Registers
**DBGCLAIMCLR**: Debug CLAIM Tag Clear register
**DBGCLAIMSET**: Debug CLAIM Tag Set register
**DBGDCCINT**: DCC Interrupt Enable Register
**DBGDEVID**: Debug Device ID register 0
**DBGDEVID1**: Debug Device ID register 1
**DBGDEVID2**: Debug Device ID register 2
**DBGIDR**: Debug ID Register
**DBGDRAR**: Debug ROM Address Register
**DBGDSAR**: Debug Self Address Register
**DBGDSCTest**: Debug Status and Control Register, External View
**DBGDSCre**: Debug Status and Control Register, Internal View
DBGDTRRxext: Debug OS Lock Data Transfer Register, Receive, External View
DBGDTRRxint: Debug Data Transfer Register, Receive
DBGDTRTXXext: Debug OS Lock Data Transfer Register, Transmit
DBGDTRTXXint: Debug Data Transfer Register, Transmit
DBGOSDLR: Debug OS Double Lock Register
DBGOSECCR: Debug OS Lock Exception Catch Control Register
DBGOSLAR: Debug OS Lock Access Register
DBGOSLSR: Debug OS Lock Status Register
DBGPRCR: Debug Power Control Register
DBGVCR: Debug Vector Catch Register
DBGWCR<n>: Debug Watchpoint Control Registers
DBGWFAR: Debug Watchpoint Fault Address Register
DBGWVR<n>: Debug Watchpoint Value Registers
DFAR: Data Fault Address Register
DFSR: Data Fault Status Register
DISR: Deferred Interrupt Status Register
DLR: Debug Link Register
DSPSR: Debug Saved Program Status Register
ELR_hyp: Exception Link Register (Hyp mode)
ERRIDR: Error Record ID Register
ERRSELR: Error Record Select Register
ERXADDR: Selected Error Record Address Register
ERXADDR2: Selected Error Record Address Register 2
ERXCTLR: Selected Error Record Control Register
ERXCTLR2: Selected Error Record Control Register 2
ERXFR: Selected Error Record Feature Register
ERXFR2: Selected Error Record Feature Register 2
ERXMISC0: Selected Error Record Miscellaneous Register 0
ERXMISC1: Selected Error Record Miscellaneous Register 1
ERXMISC2: Selected Error Record Miscellaneous Register 2
ERXMISC3: Selected Error Record Miscellaneous Register 3
ERXMISC4: Selected Error Record Miscellaneous Register 4
ERXMISC5: Selected Error Record Miscellaneous Register 5
ERXMISC6: Selected Error Record Miscellaneous Register 6
ERXMISC7: Selected Error Record Miscellaneous Register 7
ERXSTATUS: Selected Error Record Primary Status Register
FCSEIDR: FCSE Process ID register
FPESC: Floating-Point Exception Control register
FPSCR: Floating-Point Status and Control Register
FPSID: Floating-Point System ID register
HACR: Hyp Auxiliary Configuration Register
HACTLR: Hyp Auxiliary Control Register
HACTLR2: Hyp Auxiliary Control Register 2
HADFSR: Hyp Auxiliary Data Fault Status Register
HAIIFSR: Hyp Auxiliary Instruction Fault Status Register
HAMAIR0: Hyp Auxiliary Memory Attribute Indirection Register 0
HAMAIR1: Hyp Auxiliary Memory Attribute Indirection Register 1
HCPTTR: Hyp Architectural Feature Trap Register
HCR: Hyp Configuration Register
HCR2: Hyp Configuration Register 2
HDCTR: Hyp Debug Control Register
HDFAR: Hyp Data Fault Address Register
HIFAR: Hyp Instruction Fault Address Register
HMAIR0: Hyp Memory Attribute Indirection Register 0
HMAIR1: Hyp Memory Attribute Indirection Register 1
HPFAR: Hyp IPA Fault Address Register
HRMR: Hyp Reset Management Register
HSCTRL: Hyp System Control Register
HSR: Hyp Syndrome Register
HSTR: Hyp System Trap Register
HTCR: Hyp Translation Control Register
HTPIDR: Hyp Software Thread ID Register
HTRFCR: Hyp Trace Filter Control Register
HTTBR: Hyp Translation Table Base Register
HVBAR: Hyp Vector Base Address Register
ICC_AP0R<n>: Interrupt Controller Active Priorities Group 0 Registers
ICC_AP1R<n>: Interrupt Controller Active Priorities Group 1 Registers
ICC_ASGI1R: Interrupt Controller Alias Software Generated Interrupt Group 1 Register
ICC_BPR0: Interrupt Controller Binary Point Register 0
ICC_BPR1: Interrupt Controller Binary Point Register 1
ICC_CTRLR: Interrupt Controller Control Register
ICC_DIR: Interrupt Controller Deactivate Interrupt Register
ICC_EOIR0: Interrupt Controller End Of Interrupt Register 0
ICC_EOIR1: Interrupt Controller End Of Interrupt Register 1
ICC_HPPIR0: Interrupt Controller Highest Priority Pending Interrupt Register 0
ICC_HPPIR1: Interrupt Controller Highest Priority Pending Interrupt Register 1
ICC_HSRE: Interrupt Controller Hyp System Register Enable register
ICC_IAR0: Interrupt Controller Interrupt Acknowledge Register 0
ICC_IAR1: Interrupt Controller Interrupt Acknowledge Register 1
ICC_IGRPEN0: Interrupt Controller Interrupt Group 0 Enable register
ICC_IGRPEN1: Interrupt Controller Interrupt Group 1 Enable register
ICC_MCTRL: Interrupt Controller Monitor Control Register
ICC_MGRPEN1: Interrupt Controller Monitor Interrupt Group 1 Enable register
ICC_MSRE: Interrupt Controller Monitor System Register Enable register
ICC_PMR: Interrupt Controller Interrupt Priority Mask Register
ICC_RPR: Interrupt Controller Running Priority Register
ICC_SGI0R: Interrupt Controller Software Generated Interrupt Group 0 Register
ICC_SGI1R: Interrupt Controller Software Generated Interrupt Group 1 Register
ICC_SRE: Interrupt Controller System Register Enable register
ICH_AP0R<n>: Interrupt Controller Hyp Active Priorities Group 0 Registers
ICH_AP1R<n>: Interrupt Controller Hyp Active Priorities Group 1 Registers
ICH_EISR: Interrupt Controller End of Interrupt Status Register
ICH_ELBSR: Interrupt Controller Empty List Register Status Register
ICH_HCR: Interrupt Controller Hyp Control Register
ICH_L<n>: Interrupt Controller List Registers
ICH_LRC<n>: Interrupt Controller List Registers
ICH_MISR: Interrupt Controller Maintenance Interrupt State Register
ICH_VMCR: Interrupt Controller Virtual Machine Control Register
ICH_VTR: Interrupt Controller VGIC Type Register
ICV_AP0R<n>: Interrupt Controller Virtual Active Priorities Group 0 Registers
ICV_AP1R<n>: Interrupt Controller Virtual Active Priorities Group 1 Registers
ICV_BPR0: Interrupt Controller Virtual Binary Point Register 0
ICV_BPR1: Interrupt Controller Virtual Binary Point Register 1
ICV_CTRL: Interrupt Controller Virtual Control Register
ICV_DIR: Interrupt Controller Deactivate Virtual Interrupt Register
ICV_EOIR0: Interrupt Controller Virtual End Of Interrupt Register 0
ICV_EOIR1: Interrupt Controller Virtual End Of Interrupt Register 1
ICV_HPPIR0: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
**ICV_HPPIR1**: Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

**ICV_IAR0**: Interrupt Controller Virtual Interrupt Acknowledge Register 0

**ICV_IAR1**: Interrupt Controller Virtual Interrupt Acknowledge Register 1

**ICV_IGRPEN0**: Interrupt Controller Virtual Interrupt Group 0 Enable register

**ICV_IGRPEN1**: Interrupt Controller Virtual Interrupt Group 1 Enable register

**ICV_PMR**: Interrupt Controller Virtual Interrupt Priority Mask Register

**ICV_RPR**: Interrupt Controller Virtual Running Priority Register

**ID_AFR0**: Auxiliary Feature Register 0

**ID_DFR0**: Debug Feature Register 0

**ID_DFR1**: Debug Feature Register 1

**ID_ISAR0**: Instruction Set Attribute Register 0

**ID_ISAR1**: Instruction Set Attribute Register 1

**ID_ISAR2**: Instruction Set Attribute Register 2

**ID_ISAR3**: Instruction Set Attribute Register 3

**ID_ISAR4**: Instruction Set Attribute Register 4

**ID_ISAR5**: Instruction Set Attribute Register 5

**ID_ISAR6**: Instruction Set Attribute Register 6

**ID_MMFR0**: Memory Model Feature Register 0

**ID_MMFR1**: Memory Model Feature Register 1

**ID_MMFR2**: Memory Model Feature Register 2

**ID_MMFR3**: Memory Model Feature Register 3

**ID_MMFR4**: Memory Model Feature Register 4

**ID_MMFR5**: Memory Model Feature Register 5

**ID_PFR0**: Processor Feature Register 0

**ID_PFR1**: Processor Feature Register 1

**ID_PFR2**: Processor Feature Register 2

**IFAR**: Instruction Fault Address Register

**IFSR**: Instruction Fault Status Register

**ISR**: Interrupt Status Register

**JIDR**: Jazelle ID Register

**JMCR**: Jazelle Main Configuration Register

**JOSCR**: Jazelle OS Control Register

**MAIR0**: Memory Attribute Indirection Register 0

**MAIR1**: Memory Attribute Indirection Register 1

**MIDR**: Main ID Register

**MPIDR**: Multiprocessor Affinity Register
**MVBAR**: Monitor Vector Base Address Register

**MVFR0**: Media and VFP Feature Register 0

**MVFR1**: Media and VFP Feature Register 1

**MVFR2**: Media and VFP Feature Register 2

**NMRR**: Normal Memory Remap Register

**NSACR**: Non-Secure Access Control Register

**PAR**: Physical Address Register

**PMCCFILTR**: Performance Monitors Cycle Count Filter Register

**PMCCNTR**: Performance Monitors Cycle Count Register

**PMCEID0**: Performance Monitors Common Event Identification register 0

**PMCEID1**: Performance Monitors Common Event Identification register 1

**PMCEID2**: Performance Monitors Common Event Identification register 2

**PMCEID3**: Performance Monitors Common Event Identification register 3

**PMCTENCLR**: Performance Monitors Count Enable Clear register

**PMCTENSET**: Performance Monitors Count Enable Set register

**PMCR**: Performance Monitors Control Register

**PMEVCNTR<n>**: Performance Monitors Event Count Registers

**PMEVTYPER<n>**: Performance Monitors Event Type Registers

**PMINTENCLR**: Performance Monitors Interrupt Enable Clear register

**PMINTENSET**: Performance Monitors Interrupt Enable Set register

**PMMIR**: Performance Monitors Machine Identification Register

**PMOVS**: Performance Monitors Overflow Status Register

**PMOVSET**: Performance Monitors Overflow Status Set register

**PMSELR**: Performance Monitors Event Counter Selection Register

**PMSWINC**: Performance Monitors Software Increment register

**PMUSERENR**: Performance Monitors User Enable Register

**PMXEVCNTR**: Performance Monitors Selected Event Count Register

**PMXEVTPYER**: Performance Monitors Selected Event Type Register

**PRRR**: Primary Region Remap Register

**REVIDR**: Revision ID Register

**RMR**: Reset Management Register

**RVBAR**: Reset Vector Base Address Register

**SCR**: Secure Configuration Register

**SCTLR**: System Control Register

**SDCR**: Secure Debug Control Register

**SDER**: Secure Debug Enable Register
SPSR: Saved Program Status Register

SPSR_abt: Saved Program Status Register (Abort mode)

SPSR_fiq: Saved Program Status Register (FIQ mode)

SPSR_irq: Saved Program Status Register (IRQ mode)

SPSR_hyp: Saved Program Status Register (Hyp mode)

SPSR_mon: Saved Program Status Register (Monitor mode)

SPSR_svc: Saved Program Status Register (Supervisor mode)

SPSR_und: Saved Program Status Register (Undefined mode)

TCMTR: TCM Type Register

TLBTR: TLB Type Register

TPIDRPRW: PL1 Software Thread ID Register

TPIDRURO: PL0 Read-Only Software Thread ID Register

TPIDRURW: PL0 Read/Write Software Thread ID Register

TRFCR: Trace Filter Control Register

TTBCR: Translation Table Base Control Register

TTBCR2: Translation Table Base Control Register 2

TTBR0: Translation Table Base Register 0

TTBR1: Translation Table Base Register 1

VBAR: Vector Base Address Register

VDFSR: Virtual SError Exception Syndrome Register

VDISR: Virtual Deferred Interrupt Status Register

VMPIDR: Virtualization Multiprocessor ID Register

VPIDR: Virtualization Processor ID Register

VTCR: Virtualization Translation Control Register

VTTBR: Virtualization Translation Table Base Register
AArch32 System Instructions

ATS12NSOPR: Address Translate Stages 1 and 2 Non-secure Only PL1 Read
ATS12NSOPW: Address Translate Stages 1 and 2 Non-secure Only PL1 Write
ATS12NSOUR: Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read
ATS12NSOUW: Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write
ATS1CPR: Address Translate Stage 1 Current state PL1 Read
ATS1CPRP: Address Translate Stage 1 Current state PL1 Read PAN
ATS1CPW: Address Translate Stage 1 Current state PL1 Write
ATS1CPWP: Address Translate Stage 1 Current state PL1 Write PAN
ATS1CUR: Address Translate Stage 1 Current state Unprivileged Read
ATS1CUW: Address Translate Stage 1 Current state Unprivileged Write
ATS1HR: Address Translate Stage 1 Hyp mode Read
ATS1HW: Address Translate Stage 1 Hyp mode Write
BPIALL: Branch Predictor Invalidate All
BPIALLIS: Branch Predictor Invalidate All, Inner Shareable
BPIMVA: Branch Predictor Invalidate by VA
CFPRCTX: Control Flow Prediction Restriction by Context
CP15DMB: Data Memory Barrier System instruction
CP15DSB: Data Synchronization Barrier System instruction
CP15ISB: Instruction Synchronization Barrier System instruction
CPPRCTX: Cache Prefetch Prediction Restriction by Context
DCCIMVAC: Data Cache line Clean and Invalidate by VA to PoC
DCCISW: Data Cache line Clean and Invalidate by Set/Way
DCCMVAC: Data Cache line Clean by VA to PoC
DCCMVAU: Data Cache line Clean by VA to PoU
DCCSW: Data Cache line Clean by Set/Way
DCIMVAC: Data Cache line Invalidate by VA to PoC
DCISW: Data Cache line Invalidate by Set/Way
DTLBIALL: Data TLB Invalidate All
DTLBIASID: Data TLB Invalidate by ASID match
DTLBIMVA: Data TLB Invalidate by VA
DVPRCTX: Data Value Prediction Restriction by Context
ICIALLU: Instruction Cache Invalidate All to PoU
ICIALLUIS: Instruction Cache Invalidate All to PoU, Inner Shareable
ICIMVAU: Instruction Cache line Invalidate by VA to PoU
ITLBIAI: Instruction TLB Invalidate All
ITLBIAISD: Instruction TLB Invalidate by ASID match
ITLBIMVA: Instruction TLB Invalidate by VA
TLBIALL: TLB Invalidate All
TLBIALLH: TLB Invalidate All, Hyp mode
TLBIALLHIS: TLB Invalidate All, Hyp mode, Inner Shareable
TLBIALLIS: TLB Invalidate All, Inner Shareable
TLBIALLNSNH: TLB Invalidate All, Non-Secure Non-Hyp
TLBIALLNSNHIS: TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable
TLBIALS: TLB Invalidate by ASID match
TLBIALSID: TLB Invalidate by ASID match, Inner Shareable
TLBIIIPAS2: TLB Invalidate by Intermediate Physical Address, Stage 2
TLBIIIPAS2IS: TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable
TLBIIIPAS2L: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level
TLBIIIPAS2LIS: TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable
TLBIMVA: TLB Invalidate by VA
TLBIMVAA: TLB Invalidate by VA, All ASID
TLBIMVAAIS: TLB Invalidate by VA, All ASID, Inner Shareable
TLBIMVAAL: TLB Invalidate by VA, All ASID, Last level
TLBIMVAALIS: TLB Invalidate by VA, All ASID, Last level, Inner Shareable
TLBIMVAH: TLB Invalidate by VA, Hyp mode
TLBIMVAHIS: TLB Invalidate by VA, Hyp mode, Inner Shareable
TLBIMVAIS: TLB Invalidate by VA, Inner Shareable
TLBIMVAL: TLB Invalidate by VA, Last level
TLBIMVALH: TLB Invalidate by VA, Last level, Hyp mode
TLBIMVALHIS: TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable
TLBIMVALIS: TLB Invalidate by VA, Last level, Inner Shareable
ACTLR, Auxiliary Control Register

The ACTLR characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

**Configuration**

AArch32 System register ACTLR bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL1[31:0].

Some bits might define global configuration settings, and be common to the Secure and Non-secure instances of the register.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ACTLR is a 32-bit register.

**Field descriptions**

The ACTLR bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31| 30| 29| 28| 27| 26| 25| 24| 23| 22| 21| 20| 19| 18| 17| 16| 15| 14| 13| 12| 11| 10|  9|  8|  7|  6|  5|  4|  3|  2|  1|  0|
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the ACTLR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b001</td>
<td>0b000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return ACTLR_S;
        else
            return ACTLR_NS;
        end if
    else
        return ACTLR;
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ACTLR_NS;
    else
        return ACTLR;
    end if
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return ACTLR_S;
    else
        return ACTLR_NS;
    end if
MCR{<c>{q}}<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            ACTLR_S = R[t];
        else
            ACTLR_NS = R[t];
        end if
    else
        ACTLR = R[t];
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        ACTLR_NS = R[t];
    else
        ACTLR = R[t];
    end if
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ACTLR_S = R[t];
    else
        ACTLR_NS = R[t];
    end if
The ACTLR2 characteristics are:

**Purpose**

Provides additional space to the ACTLR register to hold IMPLEMENTATION DEFINED trap functionality for execution at EL1 and EL0.

**Configuration**

AArch32 System register ACTLR2 bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL1[63:32].

In Armv8.0 and Armv8.1, it is IMPLEMENTATION DEFINED whether this register is implemented, or whether it causes UNDEFINED exceptions when accessed. The implementation of this register can be detected by examining ID_MMFR4.AC2.

From Armv8.2 this register must be implemented.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ACTLR2 is a 32-bit register.

**Field descriptions**

The ACTLR2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the ACTLR2**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            ACTLR2_S = R[t];
        else
            ACTLR2_NS = R[t];
        end
    else
        ACTLR2 = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        ACTLR2_NS = R[t];
    else
        ACTLR2 = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ACTLR2_S = R[t];
    else
        ACTLR2_NS = R[t];
    end

MCR{c}<{q}> {coproc}, {#}{opc1}, <Rt>, <CRn>, <CRm>{, {#}{opc2}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
ADFSR, Auxiliary Data Fault Status Register

The ADFSR characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for Data Abort exceptions taken to EL1 modes, and EL3 modes when EL3 is implemented and is using AArch32.

**Configuration**

AArch32 System register ADFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR0_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ADFSR is a 32-bit register.

**Field descriptions**

The ADFSR bit assignments are:

<table>
<thead>
<tr>
<th>bit</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the ADFSR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return ADFSR_S;
    else
      return ADFSR_NS;
  else
    return ADFSR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return ADFSR_NS;
  else
    return ADFSR;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return ADFSR_S;
  else
    return ADFSR_NS;
else
  return ADFSR_NS;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
AIDR, Auxiliary ID Register

The AIDR characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED identification information.

The value of this register must be used in conjunction with the value of MIDR.

Configuration

AArch32 System register AIDR bits [31:0] are architecturally mapped to AArch64 System register AIDR_EL1[31:0].

Attributes

AIDR is a 32-bit register.

Field descriptions

The AIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing the AIDR

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>}} <\text{coproc}>, \{<\text{#}\text{opc1}\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\}, \{<\text{#}\text{opc2}\}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() & ELUsingAArch32(EL2) & HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() & ELUsingAArch32(EL2) & HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
elif EL2Enabled() & ELUsingAArch32(EL2) & HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() & ELUsingAArch32(EL2) & HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return AIDR;
elif PSTATE.EL == EL2 then
    return AIDR;
elif PSTATE.EL == EL3 then
    return AIDR;
AIFSR, Auxiliary Instruction Fault Status Register

The AIFSR characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED fault status information for Prefetch Abort exceptions taken to EL1 modes, and EL3 modes when EL3 is implemented and is using AArch32.

**Configuration**

AArch32 System register AIFSR bits [31:0] are architecturally mapped to AArch64 System register $AFSR1\_EL1[31:0]$.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AIFSR is a 32-bit register.

**Field descriptions**

The AIFSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 - 0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [31:0]**

This field resets to an architecturally UNKNOWN value.

**Accessing the AIFSR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \text{ <coproc>, \{#\}\text{opc1}, \text{<Rt>, <CRn>, <CRm>}, \{#\}\text{opc2}>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return AIFSR_S;
        else
            return AIFSR_NS;
        end
    else
        return AIFSR;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AIFSR_NS;
    else
        return AIFSR;
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AIFSR_S;
    else
        return AIFSR_NS;
    end
else
    return AIFSR_NS;
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            AIFSR_S = R[t];
        else
            AIFSR_NS = R[t];
        end
    else
        AIFSR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AIFSR_NS = R[t];
    else
        AIFSR = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AIFSR_S = R[t];
    else
        AIFSR_NS = R[t];
    end
else
    AIFSR_NS = R[t];
end
AMAIR0, Auxiliary Memory Attribute Indirection Register 0

The AMAIR0 characteristics are:

**Purpose**

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR0.

**Configuration**

AArch32 System register AMAIR0 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMAIR0 is a 32-bit register.

**Field descriptions**

The AMAIR0 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR0(S) gives the value for memory accesses from Secure state.
- AMAIR0(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architectured behavior specified by MAIR0 and MAIR1.

In a typical implementation, AMAIR0 and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the AMAIR0**

Accesses to this register use the following encodings:

\[ \text{MRC\{<c>\{<q>\} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2>}} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>

Page 2059
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TAKEHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TAKEHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return AMAIR0_S;
        else
            return AMAIR0_NS;
        end
    else
        return AMAIR0;
elsif PSTATE.EL == EL2 then
    if(HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIR0_NS;
    else
        return AMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AMAIR0_S;
    else
        return AMAIR0_NS;
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            AMAIR0_S = R[t];
        else
            AMAIR0_NS = R[t];
        end if;
    else
        AMAIR0 = R[t];
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIR0_NS = R[t];
    else
        AMAIR0 = R[t];
    end if;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR0_S = R[t];
        else
            AMAIR0_NS = R[t];
        end if;
    end if;
end if;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**AMAIR1, Auxiliary Memory Attribute Indirection Register 1**

The AMAIR1 characteristics are:

**Purpose**

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by MAIR1.

**Configuration**

AArch32 System register AMAIR1 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL1[63:32].

When EL3 is using AArch32, write access to AMAIR1(S) is disabled when the CP15SDISABLE signal is asserted HIGH. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMAIR1 is a 32-bit register.

**Field descriptions**

The AMAIR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR1(S) gives the value for memory accesses from Secure state.
- AMAIR1(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by MAIR0 and MAIR1.

In a typical implementation, AMAIR0 and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the AMAIR1**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return AMAIR1_S;
    else
      return AMAIR1_NS;
  else
    return AMAIR1;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return AMAIR1_NS;
  else
    return AMAIR1;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return AMAIR1_S;
  else
    return AMAIR1_NS;

MRC{c}{q} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

AMAIR1, Auxiliary Memory Attribute Indirection Register 1
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            AMAIR1_S = R[t];
        else
            AMAIR1_NS = R[t];
        end
    else
        AMAIR1 = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR1_S = R[t];
        else
            AMAIR1_NS = R[t];
        end
    end
AMCFGR, Activity Monitors Configuration Register

The AMCFGR characteristics are:

**Purpose**

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity
monitor event counters implemented, and the size of the counters. AMCFGR is applicable to both the architected and
the auxiliary counter groups.

**Configuration**

AArch32 System register AMCFGR bits [31:0] are architecturally mapped to AArch64 System register
AMCFGR_EL0[31:0].

AArch32 System register AMCFGR bits [31:0] are architecturally mapped to External register AMCFGR[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCFGR are UNDEFINED.

**Attributes**

AMCFGR is a 32-bit register.

**Field descriptions**

The AMCFGR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCG</td>
<td>RES0</td>
<td>HDBG</td>
<td>RAZ</td>
<td>SIZE</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NCG, bits [31:28]**

Defines the number of counter groups.

The number of implemented counter groups is defined as [AMCFGR.NCG + 1].

If the number of implemented auxiliary activity monitor event counters is zero, this field has a value of 0b0000. Otherwise, this field has a value of 0b0001.

**Bits [27:25]**

Reserved, RES0.

**HDBG, bit [24]**

Halt-on-debug supported.

From Armv8, this feature must be supported, and so this bit is 0b1.

<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AMCR.HDBG is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>AMCR.HDBG is read/write.</td>
</tr>
</tbody>
</table>

**Bits [23:14]**

Reserved, RAZ.
**SIZE, bits [13:8]**

Defines the size of activity monitor event counters.

The size of the activity monitor event counters implemented by the Activity Monitors Extension is defined as [AMCFGR.SIZE + 1].

From Armv8, the counters are 64-bit, and so this field is 0b111111.

---

**Note**

Software also uses this field to determine the spacing of counters in the memory-map. From Armv8, the counters are at doubleword-aligned addresses.

---

**N, bits [7:0]**

Defines the number of activity monitor event counters.

The total number of counters implemented in all groups by the Activity Monitors Extension is defined as [AMCFGR.N + 1].

---

**Accessing the AMCFGR**

Accesses to this register use the following encodings:

\[ \text{MRC\{<c>\}{<q>} <coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>\}, \{#<opc2>\}} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return AMCFGR;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return AMCFGR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return AMCFGR;
elsif PSTATE.EL == EL3 then
  return AMCFGR;
The AMCGCR characteristics are:

**Purpose**

Provides information on the number of activity monitor event counters implemented within each counter group.

**Configuration**

AArch32 System register AMCGCR bits [31:0] are architecturally mapped to AArch64 System register AMCGCR_EL0[31:0].

AArch32 System register AMCGCR bits [31:0] are architecturally mapped to External register AMCGCR[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCGCR are undefined.

**Attributes**

AMCGCR is a 32-bit register.

**Field descriptions**

The AMCGCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15-8</td>
<td>Counter Group 1 Number of Counters</td>
</tr>
<tr>
<td>7-0</td>
<td>Counter Group 0 Number of Counters</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**CG1NC, bits [15:8]**

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In AMUv1, the permitted range of values is 0 to 16.

**CG0NC, bits [7:0]**

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

In AMUv1, the value of this field is 4.

**Accessing the AMCGCR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>}{coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      else
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return AMCGCR;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR.T13 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return AMCGCR;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return AMCGCR;
  elsif PSTATE.EL == EL3 then
    return AMCGCR;
AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0 characteristics are:

**Purpose**

Disable control bits for the architected activity monitors event counters, AMEVCNTR0.<n>.

**Configuration**

AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR0_EL0[31:0].

AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to External register AMCNTENCLR0[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENCLR0 is a 32-bit register.

**Field descriptions**

The AMCNTENCLR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | P<n>, bit [n] |

**Bits [31:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter disable bit for AMEVCNTR0.<n>.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR.CG0NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR0.&lt;n&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR0.&lt;n&gt; is enabled. When written, disables AMEVCNTR0.&lt;n&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Accessing the AMCNTENCLR0**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
      elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch32.TakeHypTrapException(0x00);
      else
        UNDEFINED;
elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' then
  if EL2Enabled() && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
  AArch32.TakeHypTrapException(0x00);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
  AArch32.TakeHypTrapException(0x00);
elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return AMCNTENCLR0;
eelsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
  AArch32.TakeHypTrapException(0x00);
elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return AMCNTENCLR0;
eelsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return AMCNTENCLR0;
eelsif PSTATE.EL == EL3 then
  return AMCNTENCLR0;
if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
  AMCNTENCLR0 = R[t];
else
  UNDEFINED;
AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1 characteristics are:

**Purpose**

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>.

**Configuration**

AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR1_EL0[31:0].

AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to External register AMCNTENCLR1[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENCLR1 is a 32-bit register.

**Field descriptions**

The AMCNTENCLR1 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
| RES0 | P<n>, bit [n] |

**Bits [31:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter disable bit for AMEVCNTR1<n>.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR_EL0.CG1NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR1&lt;n&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR1&lt;n&gt; is enabled. When written, disables AMEVCNTR1&lt;n&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
Accessing the AMCNTENCLR1

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENCLR1 are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

**Note**

The number of auxiliary activity monitor event counters implemented is zero exactly when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

\[ \text{MRC\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2>} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return AMCNTENCLR1;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif IsHighestEL(PSTATE.EL) then
      AMCNTENCLR1 = R[t];
    else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return AMCNTENCLR1;
  elsif PSTATE.EL == EL3 then
    return AMCNTENCLR1;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
  AMCNTENCLR1 = R[t];
else
  UNDEFINED;
AMCNTENSET0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0 characteristics are:

**Purpose**

Enable control bits for the architected activity monitors event counters, \texttt{AMEVCNTR0<n>}. 

**Configuration**

AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register \texttt{AMCNTENSET0_EL0[31:0]}. 

AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to External register \texttt{AMCNTENSET0[31:0]}. 

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0 are UNDEFINED. 

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENSET0 is a 32-bit register.

**Field descriptions**

The AMCNTENSET0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter enable bit for \texttt{AMEVCNTR0<n>}. 

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in \texttt{AMCGCR.CG0NC}. 

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that \texttt{AMEVCNTR0&lt;n&gt;} is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that \texttt{AMEVCNTR0&lt;n&gt;} is enabled. When written, enables \texttt{AMEVCNTR0&lt;n&gt;}.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Accessing the AMCNTENSET0**

Accesses to this register use the following encodings:
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && AMUSERENR.EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() && AMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTE0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return AMCNTENSET0;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return AMCNTENSET0;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return AMCNTENSET0;
elsif PSTATE.EL == EL3 then
  return AMCNTENSET0;

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
  AMCNTENSET0 = R[t];
else
  UNDEFINED;
AMCNTENSET1, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1 characteristics are:

**Purpose**

Enable control bits for the auxiliary activity monitors event counters, AMEVCTR1<n>.

**Configuration**

AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET1_EL0[31:0].

AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to External register AMCNTENSET1[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1 are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMCNTENSET1 is a 32-bit register.

**Field descriptions**

The AMCNTENSET1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | P<n>, bit [n] |

**Bits [31:16]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 15**

Activity monitor event counter enable bit for AMEVCTR1<n>.

Bits [31:16] are RES0. Bits [15:N] are RAZ/WI. N is the value in AMCGCR.CG1NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCTR1&lt;n&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCTR1&lt;n&gt; is enabled. When written, enables AMEVCTR1&lt;n&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
Accessing the AMCNTENSET1

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENSET1 are constrained unpredictable, and the following behaviors are permitted:

- Accesses to the register are undefined.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.

Note

The number of auxiliary activity monitor counters implemented is zero when AMCFGR.NCG == 0b0000.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

AMCNTENSET1, Activity Monitors Count Enable Set Register 1
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch32.TakeHypTrapException(0x00);
        end
    else
        UNDEFINED;
    end
else
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL3) && SCR_EL3.FGTEn == '1' && (!HaveEL(EL3) ||
        SCRE2_E3.T13 == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET1;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET1;
    end
elsif PSTATE.EL == EL3 then
    return AMCNTENSET1;
end

MCR{{<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET1 = R[t];
else
    UNDEFINED;
**AMCR, Activity Monitors Control Register**

The AMCR characteristics are:

**Purpose**

Global control register for the activity monitors implementation. AMCR is applicable to both the architected and the auxiliary counter groups.

**Configuration**

AArch32 System register AMCR bits [31:0] are architecturally mapped to AArch64 System register `AMCR_EL0[31:0]`.

AArch32 System register AMCR bits [31:0] are architecturally mapped to External register `AMCR[31:0]`.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCR are UNDEFINED.

**Attributes**

AMCR is a 32-bit register.

**Field descriptions**

The AMCR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>CG1RZ</td>
<td>RES0</td>
<td>HDBG</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:18]**

Reserved, RES0.

**CG1RZ, bit [17]**

When ARMv8.6-AMU is implemented:

Counter Group 1 Read Zero.

<table>
<thead>
<tr>
<th>CG1RZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>System register reads of <code>AMEVCNTR1&lt;n&gt;</code> return the event count at all implemented and enabled Exception levels.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the current Exception level is the highest implemented Exception level, system register reads of <code>AMEVCNTR1&lt;n&gt;</code> return the event count. Otherwise, reads of <code>AMEVCNTR1&lt;n&gt;</code> return a zero value.</td>
</tr>
</tbody>
</table>

**Note**

Reads from the memory-mapped view are unaffected by this field.

**Otherwise:**

Reserved, RES0.
Bits [16:11]
Reserved, RES0.

HDBG, bit [10]
This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Activity monitors do not halt counting when the PE is halted in Debug state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Activity monitors halt counting when the PE is halted in Debug state.</td>
</tr>
</tbody>
</table>

Bits [9:0]
Reserved, RES0.

Accessing the AMCR

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE != '11' && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCR;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return AMCR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return AMCR;
    elsif PSTATE.EL == EL3 then
        return AMCR;
end if

MCR{<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    endif
else
    ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch32.TakeHypTrapException(0x00);
        endif
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
    AMCR = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.TakeHypTrapException(0x03);
    endif
else
    ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
            AArch32.TakeHypTrapException(0x03);
        else
            AMCR = R[t];
        endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
    AMCR = R[t];
elsif PSTATE.EL == EL3 then
    AMCR = R[t];

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
AMEVCNTR0<\(n\)>, Activity Monitors Event Counter Registers 0, \(n = 0 - 15\)

The AMEVCNTR0<\(n\)> characteristics are:

**Purpose**

Provides access to the architected activity monitor event counters.

**Configuration**

AArch32 System register AMEVCNTR0<\(n\)> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR0<\(n\>_EL0[63:0].

AArch32 System register AMEVCNTR0<\(n\)> bits [63:0] are architecturally mapped to External register AMEVCNTR0<\(n\> [63:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<\(n\)> are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

AMEVCNTR0<\(n\)> is a 64-bit register.

**Field descriptions**

The AMEVCNTR0<\(n\)> bit assignments are:

\[
\begin{array}{ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
Accesses to the register are UNDEFINED.
Accesses to the register behave as RAZ/WI.
Accesses to the register execute as a NOP.

Note

**AMCGCR.CG0NC** identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

\[
\text{MRRC}\{c\}\{q\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{Rt2}>, <\text{CRm}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000:n[3]</td>
<td>0b0:n[2:0]</td>
</tr>
</tbody>
</table>
if CRm == 0 then
  if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
      else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
      end if
    else
      if EL2Enabled() && !ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
      else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
      end if
    end if
  elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
    end if
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR0<n> == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
  else
    return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
  end if
else
  if PSTATE.EL == EL1 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
      return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
    end if
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
      return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
    end if
  elsif PSTATE.EL == EL3 then
    return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
  else
    UNDEFINED;
  end if
end if

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000:n[3]</td>
<td>0b0:n[2:0]</td>
</tr>
</tbody>
</table>
if CRm == 0 then
  if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif IsHighestEL(PSTATE.EL) then
    AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
  else
    UNDEFINED;
else
  UNDEFINED;
AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR1<n>_EL0[63:0].

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to External register AMEVCNTR1<n>[63:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

Attributes

AMEVCNTR1<n> is a 64-bit register.

Field descriptions

The AMEVCNTR1<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ACNT | ACNT |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If ARMv8.6-AMU is implemented, HCR_EL2.AMVOFFEN is 1, SCR_EL3.AMVOFFEN is 1, HCR_EL2 {E2H, TGE} is not {1,1}, EL2 is using AArch64 and is implemented in the current Security state, and AMCR_EL0.CG1RZ is 0, reads to these registers at EL0 or EL1 return (PCount<63:0> - AMEVCNTVOFF1<n>_EL2<63:0>).

PCount is the physical count returned when AMEVCNTR1<n> is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

Accessing the AMEVCNTR1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n> are CONstrained UNPREDICTABLE, and the following behaviors are permitted:
• Accesses to the register are UNDEFINED.
• Accesses to the register behave as RAZ/WI.
• Accesses to the register execute as a NOP.

**Note**

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b010:n[3]</td>
<td>0b0:n[2:0]</td>
</tr>
</tbody>
</table>
if CRm == 100 then
    if PSTATE.EL == EL0 then
        if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
            endif
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        endif
    endif
    elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch32.TakeHypTrapException(0x00);
        endif
    endif
    elsif ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELUsingAArch32(EL2) && HSTR.T4 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif !HighestELUsingAArch32() && AMCR_EL0.CG1RZ == '1' then
        Zeros();
    else
        return AMEVCNTR1[Int(CRm<0>:opc1<2:0>)];
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.<E2H,TGE> != '11' && AMUSERENR.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T4 == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
            endif
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        endif
    endif
    elsif !HighestELUsingAArch32() && AMCR_EL0.CG1RZ == '1' then
        Zeros();
    else
        return AMEVCNTR1[Int(CRm<0>:opc1<2:0>)];
    endif
else
    if PSTATE.EL == EL2 then
        if !HighestELUsingAArch32() && AMCR_EL0.CG1RZ == '1' then
            Zeros();
        else
            return AMEVCNTR1[Int(CRm<0>:opc1<2:0>)];
        endif
    endif
else
    if PSTATE.EL == EL3 then
        return AMEVCNTR1[Int(CRm<0>:opc1<2:0>)];
    endif
else
    if CRm == 101 then
        if PSTATE.EL == EL0 then
            if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
                if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                else
                    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
                endif
            else
                ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
else
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE != '11' && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL3) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR1<n>_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
elsif !IsHighestEL(PSTATE.EL) && !HighestELUsingAArch32() && AMCR_EL0.CG1RZ == '1' then
    Zeros();
elsif !IsHighestEL(PSTATE.EL) && HighestELUsingAArch32() && AMCR.CG1RZ == '1' then
    Zeros();
else
    return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    elsif !IsHighestEL(PSTATE.EL) && !HighestELUsingAArch32() && AMCR_EL0.CG1RZ == '1' then
        Zeros();
    elsif !IsHighestEL(PSTATE.EL) && HighestELUsingAArch32() && AMCR.CG1RZ == '1' then
        Zeros();
    else
        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    elsif !IsHighestEL(PSTATE.EL) && !HighestELUsingAArch32() && AMCR_EL0.CG1RZ == '1' then
        Zeros();
    elsif !IsHighestEL(PSTATE.EL) && HighestELUsingAArch32() && AMCR.CG1RZ == '1' then
        Zeros();
    else
        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
else
    UNDEFINED;

MCRR<coproc>{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b010:n[3]</td>
<td>0b0:n[2:0]</td>
</tr>
</tbody>
</table>
if CRm == 100 then
  if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
      else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
      else
        AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T4 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T4 == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
  else
    AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
  else
    AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
  elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
      AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
  elseif PSTATE.EL == EL3 then
    AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
  else
    CRm = 101 then
      if PSTATE.EL == EL0 then
        if !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
          if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
          else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif ELUsingAArch32(EL1) && AMUSERENR.EN == '0' then
          if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
          else
            AArch32.TakeHypTrapException(0x00);
        else
          UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
          AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
          AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
          AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
          AArch32.TakeHypTrapException(0x04);
        else
          AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
          AArch32.TakeHypTrapException(0x04);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x04);
else
  AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
else
  AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
  else
    AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
  AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
else
  UNDEFINED;

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
AMEVTYPER0<n>, Activity Monitors Event Type
Registers 0, n = 0 - 15

The AMEVTYPER0<n> characteristics are:

**Purpose**

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n> counts.

**Configuration**

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[31:0].

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to External register AMEVTYPER0<n>[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are UNDEFINED.

**Attributes**

AMEVTYPER0<n> is a 32-bit register.

**Field descriptions**

The AMEVTYPER0<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Bits [31:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

<table>
<thead>
<tr>
<th>evtCount</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0011</td>
<td>Processor frequency cycles</td>
<td>When n == 0</td>
</tr>
<tr>
<td>0x4004</td>
<td>Constant frequency cycles</td>
<td>When n == 1</td>
</tr>
<tr>
<td>0x0008</td>
<td>Instructions retired</td>
<td>When n == 2</td>
</tr>
<tr>
<td>0x4005</td>
<td>Memory stall cycles</td>
<td>When n == 3</td>
</tr>
</tbody>
</table>

**Accessing the AMEVTYPER0<n>**

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
**Note**

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings:

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b011:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAAarch32(EL1) && AMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAAarch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() && ELUsingAAarch32(EL1) && AMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAAarch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAAarch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAAarch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAarch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAAarch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAarch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAAarch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAAarch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAarch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAAarch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAarch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAAarch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAAarch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
  return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
The AMEVTYPER1<n> characteristics are:

**Purpose**

Provides information on the events that an auxiliary activity monitor event counter AMEVCTR1<n> counts.

**Configuration**

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER1<n>_EL0[31:0].

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to External register AMEVTYPER1<n>[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n> are UNDEFINED.

**Attributes**

AMEVTYPER1<n> is a 32-bit register.

**Field descriptions**

The AMEVTYPER1<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | evtCount |

**Bits [31:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCTR1<n>.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCTR1<n>, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by AMEVCTR1<n> might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter AMEVCTR1<n> is enabled, writes to this register have UNPREDICTABLE results.

**Accessing the AMEVTYPER1<n>**

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n> are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:
• Accesses to the register are **UNDEFINED**.
• Accesses to the register behave as RAZ/WI.
• Accesses to the register execute as a NOP.

**Note**

AMCGCR.CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q}> \{coproc\}, \{#\}<opc1>, \{CRn\}, \{CRm\}[^{<opc2>}]
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b111:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

```c
if (PSTATE.EL == EL0) then
    if (!ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0') then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    else
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        endif
    endif
    if (!ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                AArch32.TakeHypTrapException(0x03);
            endif
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
            endif
        endif
    else
        if PSTATE.EL == EL2 then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
            endif
        else
            if PSTATE.EL == EL3 then
                return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
            else
                if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
                else
                    return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
                endif
            endif
        endif
    endif
endif
```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b111:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
  AMEVTYPE1[UInt(CRM<0>:opc2<2:0>)] = R[t];
else
  UNDEFINED;
AMUSERENR, Activity Monitors User Enable Register

The AMUSERENR characteristics are:

**Purpose**

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR is applicable to both the architected and the auxiliary counter groups.

**Configuration**

AArch32 System register AMUSERENR bits [31:0] are architecturally mapped to AArch64 System register AMUSERENR_EL0[31:0].

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR are UNDEFINED.

**Attributes**

AMUSERENR is a 32-bit register.

**Field descriptions**

The AMUSERENR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>EN</td>
</tr>
</tbody>
</table>

**Bits [31:1]**

Reserved, RES0.

**EN, bit [0]**

Traps EL0 accesses to the activity monitors registers to EL1.

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 accesses to the activity monitors registers are trapped to EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0.</td>
</tr>
</tbody>
</table>

**Note**

- AMUSERENR can always be read at EL0 and is not governed by this bit.

**Accessing the AMUSERENR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMUSERENR;
    end if;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMUSERENR;
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMUSERENR;
    end if;
elsif PSTATE.EL == EL3 then
    return AMUSERENR;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    AMUSERENR = R[t];
  end
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAAArch32(EL2) && CPTR_EL2.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAAArch32(EL2) && HCPTR.TAM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    AMUSERENR = R[t];
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    AMUSERENR = R[t];
  end
elsif PSTATE.EL == EL3 then
  AMUSERENR = R[t];
APSR, Application Program Status Register

The APSR characteristics are:

**Purpose**

Hold program status and control information.

**Configuration**

**Attributes**

APSR is a 32-bit register.

**Field descriptions**

The APSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| N  | Z  | C  | V  | Q  | RES0 | GE  | RES0 | RES1 | RES0 |

**N, bit [31]**

Negative condition flag. Set to bit [31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

**Z, bit [30]**

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

**C, bit [29]**

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

**V, bit [28]**

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

**Q, bit [27]**

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

**Bits [26:20]**

Reserved, RES0.

**GE, bits [19:16]**

Greater than or Equal flags, for parallel addition and subtraction.
Bits [15:5]

Reserved, RES0.

Bit [4]

Reserved, RES1.

Bits [3:0]

Reserved, RES0.

It is permitted that, on a read of APSR:

- Bit[22] returns the value of PSTATE.PAN
- Bit[9] returns the value of PSTATE.E.
- Bit[4:0] returns the value of PSTATE.M[4:0]

Note

This is an exception to the general rule that an UNKNOWN field must not return information that cannot be obtained, at the current Privilege level, by an architected mechanism.

For more information see The Application Program State Register.

Accessing the APSR

APSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions. For more details, see MRS, MSR (register), and MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
The ATS12NSOPR characteristics are:

**Purpose**

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

ATS12NSOPR is a 32-bit System instruction.

**Field descriptions**

The ATS12NSOPR input value bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

**Executing the ATS12NSOPR instruction**

Accesses to this instruction use the following encodings:

\[\text{MCR}\{<c>\}{<q>} \text{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>, \{#<opc2>\}\} }\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b100</td>
</tr>
</tbody>
</table>


if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ATS12NSOPR(R[t]);
elsif PSTATE.EL == EL3 then
  ATS12NSOPR(R[t]);
ATS12NSOPW, Address Translate Stages 1 and 2 Non-secure Only PL1 Write

The ATS12NSOPW characteristics are:

**Purpose**

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if writing to the given virtual address.

**Configuration**

**Attributes**

ATS12NSOPW is a 32-bit System instruction.

**Field descriptions**

The ATS12NSOPW input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

*Input address for translation*

**Bits [31:0]**

Input address for translation. The resulting address can be read from the **PAR**.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

**Executing the ATS12NSOPW instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c}\{<q}\}\ <\text{coproc}, \{#}<\text{opc1}, <\text{Rt}, <\text{CRn}, <\text{CRm}\{, \{#}<\text{opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    ATS12NSOPW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS12NSOPW(R[t]);
ATS12NSOUR, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

The ATS12NSOUR characteristics are:

**Purpose**

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

ATS12NSOUR is a 32-bit System instruction.

**Field descriptions**

The ATS12NSOUR input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

**Executing the ATS12NSOUR instruction**

Accesses to this instruction use the following encodings:

\[ \text{MCR}\{<c>\}{<q>} \text{ <coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>{, \{#<opc2>\}}\} } \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    UNDEFINED;
  endif
elsif PSTATE.EL == EL2 then
  ATS12NSOUR(R[t]);
elsif PSTATE.EL == EL3 then
  ATS12NSOUR(R[t]);
ATS12NSOUW, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

The ATS12NSOUW characteristics are:

**Purpose**

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if writing to the given virtual address.

**Configuration**

**Attributes**

ATS12NSOUW is a 32-bit System instruction.

**Field descriptions**

The ATS12NSOUW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

**Executing the ATS12NSOUW instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  ATS12NSOUW(R[t]);
elsif PSTATE.EL == EL3 then
  ATS12NSOUW(R[t]);
ATS1CPR, Address Translate Stage 1 Current state PL1 Read

The ATS1CPR characteristics are:

**Purpose**

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

ATS1CPR is a 32-bit System instruction.

**Field descriptions**

The ATS1CPR input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Bits [31:0] Input address for translation</td>
</tr>
</tbody>
</table>

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

**Executing the ATS1CPR instruction**

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    ATS1CPR(R[t]);
elsif PSTATE.EL == EL2 then
  ATS1CPR(R[t]);
else PSTATE.EL == EL3 then
  ATS1CPR(R[t]);
ATS1CPRP, Address Translate Stage 1 Current state
PL1 Read PAN

The ATS1CPRP characteristics are:

**Purpose**

Performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a read from a location will generate a permission fault for a privileged access.

**Configuration**

This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to ATS1CPRP are UNDEFINED.

**Attributes**

ATS1CPRP is a 32-bit System instruction.

**Field descriptions**

The ATS1CPRP input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| **Input address for translation** |

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

**Executing the ATS1CPRP instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}<q> \ coproc, \ (#)\<opc1>, \ <Rt>, \ <CRn>, \ <CRm>, \ (#)\<opc2>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CPRP(R[t]);
    end
elsif PSTATE.EL == EL2 then
    ATS1CPRP(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CPRP(R[t]);
ATS1CPW, Address Translate Stage 1 Current state
PL1 Write

The ATS1CPW characteristics are:

**Purpose**

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if writing to the given virtual address.

**Configuration**

**Attributes**

ATS1CPW is a 32-bit System instruction.

**Field descriptions**

The ATS1CPW input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Input address for translation |

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

**Executing the ATS1CPW instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR} \{<c>\} \{<q>\} <\text{coproc}, \{#}\text{opc1}, <Rt>, <\text{CRn}>, <\text{CRm}\{, \{#\text{opc2}}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if \(\text{PSTATE.EL} == \text{EL0}\) then
  \text{UNDEFINED};
elsif \(\text{PSTATE.EL} == \text{EL1}\) then
  if \(\text{EL2Enabled()} \&\& \lnot \text{ELUsingAArch32(EL2)} \&\& \text{HSTR_EL2.T7} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR.T7} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  else
    \text{ATS1CPW(R[t])};
  endif
elsif \(\text{PSTATE.EL} == \text{EL2}\) then
  \text{ATS1CPW(R[t])};
elsif \(\text{PSTATE.EL} == \text{EL3}\) then
  \text{ATS1CPW(R[t])};
The ATS1CPWP characteristics are:

**Purpose**

When ARMv8.2-ATS1E1 is implemented, performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a write to the location will generate a permission fault for a privileged access.

**Configuration**

This instruction is present only when ARMv8.2-ATS1E1 is implemented. Otherwise, direct accesses to ATS1CPWP are **UNDEFINED**.

**Attributes**

ATS1CPWP is a 32-bit System instruction.

**Field descriptions**

The ATS1CPWP input value bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Input address for translation**

Input address for translation. The resulting address can be read from the **PAR**.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

**Executing the ATS1CPWP instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>, <Rt>, <\text{CRn}>, <\text{CRm}\{, \{#<\text{opc2}>}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CPWP(R[t]);
    endif
elsif PSTATE.EL == EL2 then
    ATS1CPWP(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1CPWP(R[t]);
ATS1CUR, Address Translate Stage 1 Current state
Unprivileged Read

The ATS1CUR characteristics are:

**Purpose**

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

ATS1CUR is a 32-bit System instruction.

**Field descriptions**

The ATS1CUR input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Input address for translation</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

**Executing the ATS1CUR instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CUR(R[t]);
    elsif PSTATE.EL == EL2 then
        ATS1CUR(R[t]);
    elsif PSTATE.EL == EL3 then
        ATS1CUR(R[t]);
The ATS1CUW characteristics are:

**Purpose**

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if writing to the given virtual address.

**Configuration**

**Attributes**

ATS1CUW is a 32-bit System instruction.

**Field descriptions**

The ATS1CUW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Input address for translation</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

**Executing the ATS1CUW instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1CUW(R[t]);
    endif
else
    ATS1CUW(R[t]);
endif
else
    ATS1CUW(R[t]);
endif
else
    ATS1CUW(R[t]);
endif
ATS1HR, Address Translate Stage 1 Hyp mode Read

The ATS1HR characteristics are:

**Purpose**

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if reading from the given virtual address.

**Configuration**

**Attributes**

ATS1HR is a 32-bit System instruction.

**Field descriptions**

The ATS1HR input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Input address for translation**

**Bits [31:0]**

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

**Executing the ATS1HR instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

MCR{<c>{<q>}{#<opc1>, <Rt>, <CRn>, <CRm>{, {#<opc2>}}}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

Page 2128
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHyp TrapException(0x03);
  else
    ATS1HR(R[t]);
  end if;
elsif PSTATE.EL == EL2 then
  ATS1HR(R[t]);
elsif PSTATE.EL == EL3 then
  ATS1HR(R[t]);
**ATS1HW, Address Translate Stage 1 Hyp mode Write**

The ATS1HW characteristics are:

**Purpose**

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if writing to the given virtual address.

**Configuration**

**Attributes**

ATS1HW is a 32-bit System instruction.

**Field descriptions**

The ATS1HW input value bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Input address for translation**

Bits [31:0]

Input address for translation. The resulting address can be read from the PAR.

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

**Executing the ATS1HW instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ATS1HW(R[t]);
    end if;
elsif PSTATE.EL == EL2 then
    ATS1HW(R[t]);
elsif PSTATE.EL == EL3 then
    ATS1HW(R[t]);
BPIALL, Branch Predictor Invalidate All

The BPIALL characteristics are:

**Purpose**

Invalidate all entries from branch predictors.

**Configuration**

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

**Attributes**

BPIALL is a 32-bit System instruction.

**Field descriptions**

BPIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the BPIALL instruction**

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

When HCR.FB is 1, at Non-secure EL1 this instruction executes as a BPIALLIS.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
    BPIALLIS();
  else
    BPIALL();
elsif PSTATE.EL == EL2 then
  BPIALL();
else
  BPIALL();
BPIALLIS, Branch Predictor Invalidate All, Inner Shareable

The BPIALLIS characteristics are:

**Purpose**

Invalidate all entries from branch predictors Inner Shareable.

**Configuration**

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

**Attributes**

BPIALLIS is a 32-bit System instruction.

**Field descriptions**

BPIALLIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the BPIALLIS instruction**

The PE ignores the value of \(<Rt>\). Software does not have to write a value to this register before issuing this instruction.

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <Rt>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>proc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        BPIALL();
    end
elsif PSTATE.EL == EL2 then
    BPIALL();
else
    BPIALL();

BPIMVA, Branch Predictor Invalidate by VA

The BPIMVA characteristics are:

**Purpose**

Invalidate virtual address from branch predictors.

**Configuration**

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

**Attributes**

BPIMVA is a 32-bit System instruction.

**Field descriptions**

The BPIMVA input value bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [31:0]**

Virtual address to use.

**Executing the BPIMVA instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        BPIMVA(R[t]);
    end;
elsif PSTATE.EL == EL2 then
    BPIMVA(R[t]);
elsif PSTATE.EL == EL3 then
    BPIMVA(R[t]);
CCSIDR, Current Cache Size ID Register

The CCSIDR characteristics are:

**Purpose**

Provides information about the architecture of the currently selected cache.

When ARMv8.3-CCIDX is implemented, this register is used in conjunction with CCSIDR2.

**Configuration**

AArch32 System register CCSIDR bits [31:0] are architecturally mapped to AArch64 System register CCSIDR_EL1[31:0].

The implementation includes one CCSIDR for each cache that it can access. CSSELR and the Security state select which Cache Size ID Register is accessible.

**Attributes**

CCSIDR is a 32-bit register.

**Field descriptions**

The CCSIDR bit assignments are:

**When ARMv8.3-CCIDX is implemented:**

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9  | Bit 8  | Bit 7  | Bit 6  | Bit 5  | Bit 4  | Bit 3  | Bit 2  | Bit 1  | Bit 0 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| RES0   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| Associativity |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| LineSize |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |

**Note**

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

**Bits [31:24]**

Reserved, RES0.

**Associativity, bits [23:3]**

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

**LineSize, bits [2:0]**

(Log2(Number of bytes in cache line)) - 4. For example:

For a line length of 16 bytes: Log2(16) = 4, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: Log2(32) = 5, LineSize entry = 1.
Otherwise:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| UNKNOWN | NumSets | Associativity | LineSize |

Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log₂(Number of bytes in cache line)) - 4. For example:

For a line length of 16 bytes: Log₂(16) = 4, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: Log₂(32) = 5, LineSize entry = 1.

Accessing the CCSIDR

If CSSEL.R.Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR read is treated as NOP.
- The CCSIDR read is UNDEFINED.
- The CCSIDR read returns an UNKNOWN value.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CCSIDR;
elsif PSTATE.EL == EL2 then
    return CCSIDR;
elsif PSTATE.EL == EL3 then
    return CCSIDR;
CCSIDR2, Current Cache Size ID Register 2

The CCSIDR2 characteristics are:

**Purpose**

When ARMv8.3-CCIDX is implemented, in conjunction with CCSIDR, provides information about the architecture of the currently selected cache.

When ARMv8.3-CCIDX is not implemented, this register is not implemented.

**Configuration**

AArch32 System register CCSIDR2 bits [31:0] are architecturally mapped to AArch64 System register CCSIDR2_EL1[31:0].

This register is present only when ARMv8.3-CCIDX is implemented. Otherwise, direct accesses to CCSIDR2 are UNDEFINED.

The implementation includes one CCSIDR2 for each cache that it can access. CSSEL and the Security state select which Cache Size ID Register is accessible.

**Attributes**

CCSIDR2 is a 32-bit register.

**Field descriptions**

The CCSIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>41</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>59</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>NumSets</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**NumSets, bits [23:0]**

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

**Accessing the CCSIDR2**

If CSSEL, Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR2 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR2 read is treated as NOP.
- The CCSIDR2 read is UNDEFINED.
- The CCSIDR2 read returns an UNKNOWN value.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page 2138
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CCSIDR2;
    end if
elsif PSTATE.EL == EL2 then
    return CCSIDR2;
elsif PSTATE.EL == EL3 then
    return CCSIDR2;
The CFPRCTX characteristics are:

**Purpose**

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, control flow prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

**Note**

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

**Configuration**

This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to CFPRCTX are UNDEFINED.

**Attributes**

CFPRCTX is a 32-bit System instruction.

**Field descriptions**

The CFPRCTX input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>GVMID[NS]</td>
</tr>
<tr>
<td>29</td>
<td>EL</td>
</tr>
<tr>
<td>28</td>
<td>VMID</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>GASID</td>
</tr>
<tr>
<td>25</td>
<td>ASID</td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**GVMID, bit [27]**

Execution of this instruction applies to all VMIDs or a specified VMID.

<table>
<thead>
<tr>
<th>VMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>
If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

**NS, bit [26]**

Security State.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state.</td>
</tr>
</tbody>
</table>

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

**EL, bits [25:24]**

Exception Level.

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>EL0.</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>EL3.</td>
</tr>
</tbody>
</table>

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

**VMID, bits [23:16]**

Only applies when bit[27] is 0 and either of:

- an EL1 context.
- an EL0 context when \( HCR_{EL2}.E2H==0 \) or \( HCR_{EL2}.TGE==0 \) or EL2 is using AArch32 state.

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and \( HCR_{EL2}.E2H==0 \) or \( HCR_{EL2}.TGE==0 \) or ELUsingAArch32(EL2)) then this field is treated as the current VMID.

When the instruction is executed at EL0 and \( HCR_{EL2}.E2H==1 \) and \( HCR_{EL2}.TGE==1 \) and !ELUsingAArch32(EL2)) then this field is ignored.

**Bits [15:9]**

Reserved, RES0.

**GASID, bit [8]**

Execution of this instruction applies to all ASIDs or a specified ASID.

<table>
<thead>
<tr>
<th>GASID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0, then this field is treated as 0.

**ASID, bits [7:0]**

Only applies for an EL0 context and when bit[8] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.
Executing the CFPRCTX instruction

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{c\}\{q\} \text{coproc}, \#\text{opc1}, \text{Rt}, \text{CRn}, \text{CRm}\{, \#\text{opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      UNDEFINED;
  endif
endif

elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
endif

if PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      UNDEFINED;
  endif
endif

elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  endif
else
  if PSTATE.EL == EL2 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch32.TakeHypTrapException(0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    endif
  endif
else
  if PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch32.TakeHypTrapException(0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    endif
  endif
endif

cfprctx(R[t]);

eif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      UNDEFINED;
  endif
endif

elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
endif

if PSTATE.EL == EL2 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      UNDEFINED;
  endif
endif

elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  endif
else
  if PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    endif
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch32.TakeHypTrapException(0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    endif
  endif
endif

cfprctx(R[t]);

eif PSTATE.EL == EL3 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
endif

 CFPRCTX(R[t]);
CLIDR, Cache Level ID Register

The CLIDR characteristics are:

**Purpose**

Identifies the type of cache, or caches, that are implemented at each level and can be managed using the architected cache maintenance instructions that operate by set/way, up to a maximum of seven levels. Also identifies the Level of Coherence (LoC) and Level of Unification (LoU) for the cache hierarchy.

**Configuration**

AArch32 System register CLIDR bits [31:0] are architecturally mapped to AArch64 System register `CLIDR_EL1[31:0]`.

**Attributes**

CLIDR is a 32-bit register.

**Field descriptions**

The CLIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICB</td>
<td>LoUU</td>
<td>LoC</td>
<td>LoUIS</td>
<td>Ctype7</td>
<td>Ctype6</td>
<td>Ctype5</td>
<td>Ctype4</td>
<td>Ctype3</td>
<td>Ctype2</td>
<td>Ctype1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ICB, bits [31:30]**

Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.

The possible values are:

<table>
<thead>
<tr>
<th>ICB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not disclosed by this mechanism.</td>
</tr>
<tr>
<td>0b01</td>
<td>L1 cache is the highest Inner Cacheable level.</td>
</tr>
<tr>
<td>0b10</td>
<td>L2 cache is the highest Inner Cacheable level.</td>
</tr>
<tr>
<td>0b11</td>
<td>L3 cache is the highest Inner Cacheable level.</td>
</tr>
</tbody>
</table>

**LoUU, bits [29:27]**

Level of Unification Uniprocessor for the cache hierarchy.

**LoC, bits [26:24]**

Level of Coherence for the cache hierarchy.

**LoUIS, bits [23:21]**

Level of Unification Inner Shareable for the cache hierarchy.

**Ctype<n>, bits [3(n-1)+2:3(n-1)], for n = 1 to 7**

Cache Type fields. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy. Possible values of each field are:
### Ctype<n>

<table>
<thead>
<tr>
<th>Ctype&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>No cache.</td>
</tr>
<tr>
<td>0b001</td>
<td>Instruction cache only.</td>
</tr>
<tr>
<td>0b010</td>
<td>Data cache only.</td>
</tr>
<tr>
<td>0b011</td>
<td>Separate instruction and data caches.</td>
</tr>
<tr>
<td>0b100</td>
<td>Unified cache.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If software reads the Cache Type fields from Ctype1 upwards, once it has seen a value of 000, no caches that can be managed using the architected cache maintenance instructions that operate by set/way exist at further-out levels of the hierarchy. So, for example, if Ctype3 is the first Cache Type field with a value of 000, the values of Ctype4 to Ctype7 must be ignored.

## Accessing the CLIDR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
     AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
     AArch32.TakeHypTrapException(0x03);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
     AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
     AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
     AArch32.TakeHypTrapException(0x03);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
     AArch32.TakeHypTrapException(0x03);
   else
     return CLIDR;
   endif;
elsif PSTATE.EL == EL2 then
   return CLIDR;
elsif PSTATE.EL == EL3 then
   return CLIDR;
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cb6e3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
CNTFRQ, Counter-timer Frequency register

The CNTFRQ characteristics are:

### Purpose

This register is provided so that software can discover the frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

### Configuration

AArch32 System register CNTFRQ bits [31:0] are architecturally mapped to AArch64 System register CNTFRQ_EL0[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

CNTFRQ is a 32-bit register.

### Field descriptions

The CNTFRQ bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| **Clock frequency** |

#### Bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

This field resets to an architecturally UNKNOWN value.

### Accessing the CNTFRQ

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
        CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' && CNTKCTL.PL0VCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    else
        return CNTFRQ;
    elsif PSTATE.EL == EL1 then
        return CNTFRQ;
    elsif PSTATE.EL == EL2 then
        return CNTFRQ;
    elsif PSTATE.EL == EL3 then
        return CNTFRQ;
else
    if IsHighestEL(PSTATE.EL) then
        CNTFRQ = R[t];
    else
        UNDEFINED;
    return CNTFRQ;
end if

MCR{<c>}{<q>}{<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b0000</td>
<td>0b1110</td>
<td>0b0000</td>
</tr>
</tbody>
</table>

if IsHighestEL(PSTATE.EL) then
    CNTFRQ = R[t];
else
    UNDEFINED;
The CNTHCTL characteristics are:

**Purpose**

Controls the generation of an event stream from the physical counter, and access from Non-secure EL1 modes to the physical counter and the Non-secure EL1 physical timer.

**Configuration**

AArch32 System register CNTHCTL bits [31:0] are architecturally mapped to AArch64 System register CNTHCTL_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHCTL is a 32-bit register.

**Field descriptions**

The CNTHCTL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | EVNTIS | RES0 | EVNTI | EVNTDIR | EVNTEN | PL1PCEN | PL1PCTEN |

**Bits [31:18]**

Reserved, RES0.

**EVNTIS, bit [17]**

When ARMv8.6-ECV is implemented:

Controls the scale of the generation of the event stream.

<table>
<thead>
<tr>
<th>EVNTIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CNTHCTL.EVNTI field applies to CNTPCT[15:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>The CNTHCTL.EVNTI field applies to CNTPCT[23:8].</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [16:8]**

Reserved, RES0.
EVNTI, bits [7:4]

Selects which bit (0 to 15) of the counter register \texttt{CNTPCT} is the trigger for the event stream generated from that counter, when that stream is enabled.

This field resets to an architecturally \texttt{UNKNOWN} value.

EVNTDIR, bit [3]

Controls which transition of the counter register \texttt{CNTPCT} trigger bit, defined by EVNTI, generates an event when the event stream is enabled:

<table>
<thead>
<tr>
<th>EVNTDIR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A 0 to 1 transition of the trigger bit triggers an event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A 1 to 0 transition of the trigger bit triggers an event.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally \texttt{UNKNOWN} value.

EVNTE, bit [2]

Enables the generation of an event stream from the counter register \texttt{CNTPCT}:

<table>
<thead>
<tr>
<th>EVNTE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables the event stream.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables the event stream.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally \texttt{UNKNOWN} value.

PL1PCEN, bit [1]

Traps Non-secure EL0 and EL1 accesses to the physical timer registers to Hyp mode:

<table>
<thead>
<tr>
<th>PL1PCEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL0 and EL1 accesses to the \texttt{CNTP_CTL}, \texttt{CNTP_CVAL}, and \texttt{CNTP_TVAL} are trapped to Hyp mode, unless the it is trapped by \texttt{CNTKCTL.PL0PTEN}.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

This field resets to an architecturally \texttt{UNKNOWN} value.

PL1PCTEN, bit [0]

Traps Non-secure EL0 and EL1 accesses to the physical counter register to Hyp mode:

<table>
<thead>
<tr>
<th>PL1PCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL0 and EL1 accesses to the \texttt{CNTPCT} are trapped to Hyp mode, unless it is trapped by \texttt{CNTKCTL.PL0PCTEN}.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

This field resets to an architecturally \texttt{UNKNOWN} value.

Accessing the CNTHCTL

Accesses to this register use the following encodings:
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  return CNTHCTL;
elsif PSTATE.EL == EL3 then
  return CNTHCTL;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  CNTHCTL = R[t];
elsif PSTATE.EL == EL3 then
  CNTHCTL = R[t];
The CNTHP_CTL characteristics are:

### Purpose

Control register for the Hyp mode physical timer.

### Configuration

AArch32 System register CNTHP_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHP_CTL_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

CNTHP_CTL is a 32-bit register.

### Field descriptions

The CNTHP_CTL bit assignments are:

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>RES0</th>
<th>ISTATUS</th>
<th>IMASK</th>
<th>ENABLE</th>
</tr>
</thead>
</table>

#### Bits [31:3] RES0

Reserved, RES0.

#### ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see ‘Operation of the CompareValue views of the timers’ and ‘Operation of the TimerValue views of the timers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

#### IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:
**IMASK, bit [4]**

Meaning for IMASK:
- **0b0**: Timer interrupt is not masked by the IMASK bit.
- **0b1**: Timer interrupt is masked by the IMASK bit.

For more information, see the description of the ISTATUS bit.

In a system where the PE resets into EL2 or EL3, this field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTHP_TVAL** continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**Accessing the CNTHP_CTL**

Accesses to this register use the following encodings:

\[
\text{MRC} \{<c>\} \{<q>\} \text{<coproc>}, \{#\text{<opc1>}, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{#\text{<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

\[
\text{MCR} \{<c>\} \{<q>\} \text{<coproc>}, \{#\text{<opc1>}, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{#\text{<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL0 then}
\text{ UNDEFINED;}
\text{elsif PSTATE.EL == EL1 then}
\text{ UNDEFINED;}
\text{elsif PSTATE.EL == EL2 then}
\text{ return CNTHP_CTL;}
\text{elsif PSTATE.EL == EL3 then}
\text{ return CNTHP_CTL;}
\]

\[
\text{if PSTATE.EL == EL0 then}
\text{ UNDEFINED;}
\text{elsif PSTATE.EL == EL1 then}
\text{ UNDEFINED;}
\text{elsif PSTATE.EL == EL2 then}
\text{ CNTHP_CTL = R[t];}
\text{elsif PSTATE.EL == EL3 then}
\text{ CNTHP_CTL = R[t];}
\]
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
else
  EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  else
    return CNTP_CTL_EL2;
else
  EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTP_CTL_SH;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    return CNTP_CTL_NS;
  else
    return CNTP_CTL;
else
  return CNTP_CTL_NS;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return CNTP_CTL_S;
    else
      return CNTP_CTL_NS;
  else
    return CNTP_CTL;
else
  return CNTP_CTL_NS;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return CNTP_CTL_NS;
  else
    return CNTP_CTL_NS;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return CNTP_CTL_S;
  else
    return CNTP_CTL_NS;
else
  return CNTP_CTL_NS;
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  endif
endif

if PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
endif

if PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL3) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
    CNTHP_CTL_EL2 = R[t];
  else
    CNTHP_CTL_EL2 = R[t];
  endif
endif

if PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    CNTP_CTL_S = R[t];
  else
    CNTP_CTL_NS = R[t];
  endif
else
  CNTP_CTL = R[t];
endif
The CNTHP_CVAL characteristics are:

### Purpose

Holds the compare value for the Hyp mode physical timer.

### Configuration

AArch32 System register CNTHP_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHP_CVAL_EL2[63:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

CNTHP_CVAL is a 64-bit register.

### Field descriptions

The CNTHP_CVAL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| **CompareValue** |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| **CompareValue** |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

### CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When CNTHP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHP_CTL.ISTATUS is set to 1.
- If CNTHP_CTL.IMASK is 0, an interrupt is generated.

When CNTHP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

### Accessing the CNTHP_CVAL

Accesses to this register use the following encodings:

MRRC{<c>}{<q>}{<coproc>}, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CVAL;
elsif PSTATE.EL == EL3 then
    return CNTHP_CVAL;

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL = R[t2]:R[t];

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !EL2Enabled() && HCR_EL2.<E2H,TGE> == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' then
        AArch32.TakeHypTrapException(0x00);
    else
        UNDEFINED;
    else
        return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL;
    else
        return CNTP_CVAL;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                return CNTP_CVAL_S;
            else
                return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            return CNTP_CVAL_S;
        else
            return CNTP_CVAL_NS;
    else
        return CNTP_CVAL_NS;

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    else
        AArch64.AArch32SystemAccessTrap(EL0, 0x04);
    endif
else
    ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
else
    EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
    EL2Enabled() && ELUsingAArch32(EL2) && HCR.EL2.<E2H,TGE> == '1' then
        AArch32.TakeHypTrapException(0x00);
else
    UNDEFINED;
endif
else
    EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHTCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHTCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHTCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTHTCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        UNDEFINED;
    endif
endif
else
    EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '0' then
        CNTHPS_CVAL_EL2 = R[t2]:R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '1' then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
    endif
else
    PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHTCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        endif
    else
        EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHTCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        endif
    endif
else
    EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
        AArch32.TakeHypTrapException(0x00);
    else
        HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                CNTP_CVAL_S = R[t2]:R[t];
            else
                CNTP_CVAL_NS = R[t2]:R[t];
            endif
        else
            CNTP_CVAL = R[t2]:R[t];
        endif
    endif
else
    EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    endif
else
    HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_CVAL_S = R[t2]:R[t];
        else
            CNTP_CVAL_NS = R[t2]:R[t];
        endif
    else
        CNTP_CVAL = R[t2]:R[t];
    endif
else
    SCR.NS == '0' then
        CNTP_CVAL_S = R[t2]:R[t];
    else
        CNTP_CVAL_NS = R[t2]:R[t];
    endif
The CNTHP_TVAL characteristics are:

**Purpose**

Holds the timer value for the Hyp mode physical timer.

**Configuration**

AArch32 System register CNTHP_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHP_TVAL_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHP_TVAL is a 32-bit register.

**Field descriptions**

The CNTHP_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | TimerValue |

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If CNTHP_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHP_CTL.ENABLE is 1, the value returned is (CNTHP_CVAL - CNTPCT).

On a write of this register, CNTHP_CVAL is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CNTHP_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHP_CTL.ISTATUS is set to 1.
- If CNTHP_CTL.IMASK is 0, an interrupt is generated.

When CNTHP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTHP_TVAL**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_TVAL;
elsif PSTATE.EL == EL3 then
    return CNTHP_TVAL;

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    CNTHP_TVAL = R[t];

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    CNTHP_TVAL = R[t];

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    CNTHP_TVAL = R[t];
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    return CNTHPS_TVAL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    return CNTHP_TVAL_EL2;
  else
    return CNTP_TVAL;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
      if SCR.NS == '0' then
        return CNTP_TVAL_S;
      else
        return CNTP_TVAL_NS;
    else
      return CNTP_TVAL;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
      return CNTP_TVAL_NS;
  else
    return CNTP_TVAL;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return CNTP_TVAL_S;
    else
      return CNTP_TVAL_NS;
  else
    return CNTP_TVAL_NS;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.E2H,TGE == '1') && CNTKTL_EL1.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
else ELUsingAArch32(EL1) && CNTKTL.PL0PTEN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
endif
elsif ELUsingAArch32(EL1) && CNTKTL.PL0PTEN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL.PL1PCEN == '0' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
  CNTHPS_TVAL_EL2 = R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
  CNTHP_TVAL_EL2 = R[t];
else
  CNTP_TVAL = R[t];
endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
else
  if SCR.NS == '0' then
    CNTP_TVAL_S = R[t];
  else
    CNTP_TVAL_NS = R[t];
  endif
else
  CNTP_TVAL = R[t];
endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    CNTP_TVAL_NS = R[t];
  else
    CNTP_TVAL = R[t];
  endif
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    CNTP_TVAL_S = R[t];
  else
    CNTP_TVAL_NS = R[t];
  endif
endif
The CNTHPS_CTL characteristics are:

### Purpose

Provides AArch32 access to the Secure EL2 physical timer.

### Note

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is \{1, 0\}.

### Configuration

AArch32 System register CNTHPS_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHPS_CTL_EL2[31:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_CTL are undefined.

RW fields in this register reset to architecturally unknown values.

### Attributes

CNTHPS_CTL is a 32-bit register.

### Field descriptions

The CNTHPS_CTL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>ISTATUS</td>
</tr>
<tr>
<td>1</td>
<td>IMASK</td>
</tr>
<tr>
<td>0</td>
<td>ENABLE</td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the CNTHPS_CTL.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL.ENABLE bit is 0, the ISTATUS field is unknown.

For more information see ‘Operation of the CompareValue views of the timers’ and ‘Operation of the TimerValue views of the timers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.
This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTHPS_TVAL_EL2** continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTHPS_CTL**

This register is accessed using the encoding for **CNTP_CTL**.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \text{ <coproc>, (#)<opc1>, <Rt>, <CRn>, <CRm>{, (#)<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !EL2Enabled() && HCR_EL2.<E2H,TGE> == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    endif
else
    ELUsingAArch32(EL1) && CNTKCTL_EL1.ELOPTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        else
            UNDEFINED;
        endif
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        if CNTKCTL_EL1.EL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                if EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL1.TGE == '1' then
                    AArch32.TakeHypTrapException(0x00);
                else
                    UNDEFINED;
                endif
            endif
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTKCTL_EL2.EL1PCEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTKCTL_EL2.EL0PTEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
                return CNTP_CTL_EL2;
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
                return CNTHP_CTL_EL2;
            else
                return CNTP_CTL;
            endif
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTKCTL_EL2.EL1PCEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTKCTL_EL2.EL0PTEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                if EL2Enabled() && ELUsingAArch32(EL2) && CNTKCTL.PL1PCEN == '0' then
                    AArch32.TakeHypTrapException(0x03);
                else
                    if SCR.NS == '0' then
                        return CNTP_CTL_S;
                    else
                        return CNTP_CTL_NS;
                    endif
                endif
            endif
        else
            if HaveEL(EL3) && ELUsingAArch32(EL3) then
                if SCR.NS == '0' then
                    return CNTP_CTL_S;
                else
                    return CNTP_CTL_NS;
                endif
            else
                return CNTP_CTL;
            endif
        endif
    endif
else
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
    endif
else
    if SCR.NS == '0' then
        return CNTP_CTL_S;
    else
        return CNTP_CTL_NS;
    endif
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch32.TakeHypTrapException(0x00);
  endif
else
  UNDEFINED;
endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '0' && CNTKCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    CNTHPS_CTL_EL2 = R[t];
  else
    CNTHPS_CTL_EL2 = R[t];
  endif
else
  if EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' && CNTKCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
endif
if PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTKCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
endif
if PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      CNTP_CTL_S = R[t];
    else
      CNTP_CTL_NS = R[t];
    endif
  else
    CNTP_CTL = R[t];
  endif
else
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      CNTP_CTL_S = R[t];
    else
      CNTP_CTL_NS = R[t];
    endif
  else
    CNTP_CTL = R[t];
  endif
else
  if SCR.NS == '0' then
    CNTP_CTL_S = R[t];
  else
    CNTP_CTL_NS = R[t];
  endif
else
  CNTP_CTL = R[t];
endif

**CNTHPS_CVAL, Counter-timer Secure Physical Timer CompareValue Register (EL2)**

The CNTHPS_CVAL characteristics are:

**Purpose**

Provides AArch32 access to the compare value for the Secure EL2 physical timer.

**Note**

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of SCR_EL3.{EEL2, NS} is {1, 0}.

**Configuration**

AArch32 System register CNTHPS_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHPS_CVAL_EL2[63:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_CVAL are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHPS_CVAL is a 64-bit register.

**Field descriptions**

The CNTHPS_CVAL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    | CompareValue |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**CompareValue, bits [63:0]**

Holds the EL2 physical timer CompareValue.

When CNTHPS_CTL_EL2.ENABLE is 1, the timer condition is met when (CNTPCT_EL0 - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHPS_CTL_EL2.ISTATUS is set to 1.
- If CNTHPS_CTL_EL2.IMASK is 0, an interrupt is generated.

When CNTHPS_CTL_EL2.ENABLE is 0, the timer condition is not met, but CNTPCT_EL0 continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.
Accessing the CNTHPS_CVAL

This register is accessed using the encoding for CNTP_CVAL.

Accesses to this register use the following encodings:

\[
\text{MRRC}{(<c>}{<q>}) \text{ proc, } \{#\text{opc1}\}, \text{ <Rt>, <Rt2>, <CRm>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if ElEnabled() && !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                return CNTP_CVAL_S;
            else
                return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CVAL_NS;
        else
            return CNTP_CVAL_NS;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            return CNTP_CVAL_S;
        else
            return CNTP_CVAL_S;
    else
        return CNTP_CVAL_NS;

MCRR{<coproc>}{<q>}{<CRm>}{#<opc1>}, <Rt>, <Rt2>, <CRm>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  endif
elsif EL2Enabled() && CNTKCTL.PL0PTEN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
  AArch32.TakeHypTrapException(0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
  CNTHPS_CVAL_EL2 = R[t2]:R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
  CNTHP_CVAL_EL2 = R[t2]:R[t];
else
  CNTP_CVAL = R[t2]:R[t];
endif
elsif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x04);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      CNTP_CVAL_S = R[t2]:R[t];
    else
      CNTP_CVAL_NS = R[t2]:R[t];
    endif
  else
    CNTP_CVAL = R[t2]:R[t];
  endif
elsif PSTATE.EL == EL2 then
  if !ELUsingAArch32(EL3) && SCR.NS == '0' then
    CNTP_CVAL_NS = R[t2]:R[t];
  else
    CNTP_CVAL = R[t2]:R[t];
  endif
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    CNTP_CVAL_S = R[t2]:R[t];
  else
    CNTP_CVAL_NS = R[t2]:R[t];
  endif
endif
The CNTHPS_TVAL characteristics are:

**Purpose**

Provides AArch32 access to the timer value for the Secure EL2 physical timer.

**Note**

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of \( \text{SCR EL3} \) \{EEL2, NS\} is \{1, 0\}.

**Configuration**

AArch32 System register CNTHPS_TVAL bits \[31:0\] are architecturally mapped to AArch64 System register \( \text{CNTHPS_TVAL EL2}[31:0] \).

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHPS_TVAL are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTHPS_TVAL is a 32-bit register.

**Field descriptions**

The CNTHPS_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If \( \text{CNTHPS_CTL_EL2 ENABLE} \) is 0, the value returned is UNKNOWN.
- If \( \text{CNTHPS_CTL_EL2 ENABLE} \) is 1, the value returned is \( (\text{CNTHPS_CVAL_EL2} - \text{CNTPCT_EL0}) \).

On a write of this register, \( \text{CNTHPS_CVAL_EL2} \) is set to \( (\text{CNTPCT_EL0} + \text{TimerValue}) \), where TimerValue is treated as a signed 32-bit integer.

When \( \text{CNTHPS_CTL_EL2 ENABLE} \) is 1, the timer condition is met when \( (\text{CNTPCT_EL0} - \text{CNTHPS_CVAL_EL2}) \) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- \( \text{CNTHPS_CTL_EL2 ISTATUS} \) is set to 1.
- If \( \text{CNTHPS_CTL_EL2 IMASK} \) is 0, an interrupt is generated.

When \( \text{CNTHPS_CTL_EL2 ENABLE} \) is 0, the timer condition is not met, but \( \text{CNTPCT_EL0} \) continues to count, so the TimerValue view appears to continue to count down.
This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTHPS_TVAL**

This register is accessed using the encoding for **CNTP_TVAL**.

Accesses to this register use the following encodings:

\[ \text{MRC}\{<c>\}{<q>}\ <\text{coproc}, \{#\}<\text{opc1}, \ <Rt>, \ <CRn>, \ <CRm}\{, \{#\}<\text{opc2}\} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    endif
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    endif
endif
endif
elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
        AArch32.TakeHypTrapException(0x00);
    else
        UNDEFINED;
    endif
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    endif
endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHPS_TVAL_EL2;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHP_TVAL_EL2;
else
    return CNTP_TVAL;
endif
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHPS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_TVAL_EL2;
    else
        return CNTP_TVAL;
    endif
else
    return CNTP_TVAL;
endif
elsif PSTATE_EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_TVAL_NS;
    else
        return CNTP_TVAL;
    endif
else
    return CNTP_TVAL;
endif
elsif PSTATE_EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_TVAL_S;
    else
        return CNTP_TVAL_NS;
    endif
else
    return CNTP_TVAL;
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    end if
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
    end if
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    CNTHPS_TVAL_EL2 = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    CNTHP_TVAL_EL2 = R[t];
  else
    CNTP_TVAL = R[t];
  end if
else
  CNTP_TVAL = R[t];
elsif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      CNTP_TVAL_S = R[t];
    else
      CNTP_TVAL_NS = R[t];
    end if
  else
    CNTP_TVAL = R[t];
  end if
else
  CNTP_TVAL = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    CNTP_TVAL_NS = R[t];
  else
    CNTP_TVAL = R[t];
  end if
else
  CNTP_TVAL = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    CNTP_TVAL_S = R[t];
  else
    CNTP_TVAL_NS = R[t];
  end if
else
  CNTP_TVAL = R[t];
end if
CNTHV_CTL, Counter-timer Virtual Timer Control register (EL2)

The CNTHV_CTL characteristics are:

**Purpose**

Provides AArch32 access to the control register for the EL2 virtual timer.

**Note**

The EL2 virtual timer is implemented by ARMv8.1-VHE. It is only accessible from AArch32 state when EL0 is using AArch32, EL2 is using AArch64, and the value of HCR_EL2.(E2H, TGE) is {1, 1}.

**Configuration**

AArch32 System register CNTHV_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTHV_CTL_EL2[31:0].

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CNTHV_CTL are UNDEFINED.

**Attributes**

CNTHV_CTL is a 32-bit register.

**Field descriptions**

The CNTHV_CTL bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>ISTATUS</td>
</tr>
<tr>
<td>1</td>
<td>IMASK</td>
</tr>
<tr>
<td>0</td>
<td>ENABLE</td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.
**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHV_TVAL continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

**Accessing the CNTHV_CTL**

This register is accessed using the encoding for CNTHV_CTL.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#<\text{opc2}>\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHV_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
    endif
elsif PSTATE.EL == EL1 then
    return CNTV_CTL;
elsif PSTATE.EL == EL2 then
    return CNTV_CTL;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL;
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch32.TakeHypTrapException(0x00);
  end;
elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
  CNTHV_CTL_EL2 = R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
  CNTHV_CTL_EL2 = R[t];
else
  CNTV_CTL = R[t];
eelsif PSTATE.EL == EL1 then
  CNTV_CTL = R[t];
eelsif PSTATE.EL == EL2 then
  CNTV_CTL = R[t];
eelsif PSTATE.EL == EL3 then
  CNTV_CTL = R[t];
The CNTHV_CVAL characteristics are:

**Purpose**

Provides AArch32 access to the compare value for the EL2 virtual timer.

**Note**

The EL2 virtual timer is implemented by ARMv8.1-VHE. It is only accessible from AArch32 state when EL0 is using AArch32, EL2 is using AArch64, and the value of HCR_EL2.{E2H, TGE} is {1, 1}.

**Configuration**

AArch32 System register CNTHV_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHV_CVAL_EL2[63:0]. This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CNTHV_CVAL are UNDEFINED.

**Attributes**

CNTHV_CVAL is a 64-bit register.

**Field descriptions**

The CNTHV_CVAL bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**CompareValue, bits [63:0]**

Holds the EL2 virtual timer CompareValue.

When CNTHV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHV_CTL.ISTATUS is set to 1.
- If CNTHV_CTL.IMASK is 0, an interrupt is generated.

When CNTHV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

**Accessing the CNTHV_CVAL**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  elsif EL2Enabled() && CNTKCTL.PL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    return CNTHV_CVAL_EL2;  
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHV_CVAL_EL2;
  else
    return CNTV_CVAL;
  end;
elsif PSTATE.EL == EL1 then
  return CNTV_CVAL;
elsif PSTATE.EL == EL2 then
  return CNTV_CVAL;
elsif PSTATE.EL == EL3 then
  return CNTV_CVAL;
else
  return CNTV_CVAL;
end;

MCRR{<c>{<q>}
<coproc>, {#<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0011</td>
</tr>
</tbody>
</table>

MRRC{<c>{<q>}
<coproc>, {#<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_CVAL_EL2 = R[t2]:R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = R[t2]:R[t];
    else
        CNTV_CVAL = R[t2]:R[t];
    elsif PSTATE.EL == EL1 then
        CNTV_CVAL = R[t2]:R[t];
    elsif PSTATE.EL == EL2 then
        CNTV_CVAL = R[t2]:R[t];
    elsif PSTATE.EL == EL3 then
        CNTV_CVAL = R[t2]:R[t];
The CNTHV_TVAL characteristics are:

**Purpose**

Provides AArch32 access to the timer value for the EL2 virtual timer.

---

**Note**

The EL2 virtual timer is implemented by ARMv8.1-VHE. It is only accessible from AArch32 state when EL0 is using AArch32, EL2 is using AArch64, and the value of HCR_EL2.(E2H, TGE) is {1, 1}.

---

**Configuration**

AArch32 System register CNTHV_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTHV_TVAL_EL2[31:0].

This register is present only when ARMv8.1-VHE is implemented. Otherwise, direct accesses to CNTHV_TVAL are UNDEFINED.

**Attributes**

CNTHV_TVAL is a 32-bit register.

**Field descriptions**

The CNTHV_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | TimerValue

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If CNTHV_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTHV_CTL.ENABLE is 1, the value returned is (CNTHV_CVAL - CNTVCT).

On a write of this register, CNTHV_CVAL is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTHV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CNTHV_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTHV_CTL.ISTATUS is set to 1.
- If CNTHV_CTL.IMASK is 0, an interrupt is generated.

When CNTHV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count, so the TimerValue view appears to continue to count down.
Accessing the CNTHV_TVAL

This register is accessed using the encoding for CNTV_TVAL.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHVS_TVAL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHV_TVAL_EL2;
  else
    return CNTV_TVAL;
  end
elsif PSTATE.EL == EL1 then
  return CNTV_TVAL;
elsif PSTATE.EL == EL2 then
  return CNTV_TVAL;
elsif PSTATE.EL == EL3 then
  return CNTV_TVAL;
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    CNTHVS_TVAL_EL2 = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    CNTHV_TVAL_EL2 = R[t];
  else
    CNTV_TVAL = R[t];
  end
elsif PSTATE.EL == EL1 then
  CNTV_TVAL = R[t];
elsif PSTATE.EL == EL2 then
  CNTV_TVAL = R[t];
elsif PSTATE.EL == EL3 then
  CNTV_TVAL = R[t];
The CNTHVS_CTL characteristics are:

**Purpose**

Provides AArch32 access to the Secure EL2 virtual timer.

**Note**

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of `SCR_EL3.{EEL2, NS}` is {1, 0}.

**Configuration**

AArch32 System register CNTHVS_CTL bits [31:0] are architecturally mapped to AArch64 System register `CNTHVS_CTL_EL2[31:0]`.

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_CTL are UNDEFINED.

**Attributes**

CNTHVS_CTL is a 32-bit register.

**Field descriptions**

The CNTHVS_CTL bit assignments are:

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>RES0</th>
<th>ISTATUS</th>
<th>IMASK</th>
<th>ENABLE</th>
</tr>
</thead>
</table>

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D8.

This bit is read-only.
**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from CNTHVS_TVAL continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

**Accessing the CNTHVS_CTL**

This register is accessed using the encoding for CNTV_CTL.

Accesses to this register use the following encodings:

\[ \text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#\}<\text{opc2}>\} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
    elsif PSTATE.EL == EL1 then
        return CNTV_CTL;
    elsif PSTATE.EL == EL2 then
        return CNTV_CTL;
    elsif PSTATE.EL == EL3 then
        return CNTV_CTL;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        endif
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        endif
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
    endif
elseif PSTATE.EL == EL1 then
    CNTV_CTL = R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];
endif

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The CNTHVS_CVAL characteristics are:

**Purpose**

Provides AArch32 access to the compare value for the Secure EL2 virtual timer.

**Note**

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of {SCR_EL3.EEL2, NS} is {1, 0}.

**Configuration**

AArch32 System register CNTHVS_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTHVS_CVAL_EL2[63:0].

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_CVAL are undefined.

**Attributes**

CNTHVS_CVAL is a 64-bit register.

**Field descriptions**

The CNTHVS_CVAL bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>CompareValue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CompareValue</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CompareValue, bits [63:0]**

Holds the EL2 virtual timer CompareValue.

When CNTHVS_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTHVS_CTL.ISTATUS is set to 1.
- If CNTHVS_CTL.IMASK is 0, an interrupt is generated.

When CNTHVS_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

**Accessing the CNTHVS_CVAL**

This register is accessed using the encoding for CNTV_CVAL.
Accesses to this register use the following encodings:

\[
\text{MRRC}\{<c>}\{<q}\} \ <\text{coproc}\>, \ {#}<\text{opc1}\>, \ <\text{Rt}\>, \ <\text{Rt2}\>, \ <\text{CRm}\>
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{coproc} & \text{CRm} & \text{opc1} \\
\hline
0b1111 & 0b1110 & 0b0011 \\
\hline
\end{array}
\]

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHVS_CVAL_EL2;
  else EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHV_CVAL_EL2;
  else
    return CNTV_CVAL;
else PSTATE.EL == EL1 then
  return CNTV_CVAL;
else PSTATE.EL == EL2 then
  return CNTV_CVAL;
else PSTATE.EL == EL3 then
  return CNTV_CVAL;

\[
\text{MCRR}\{<c>\}{<q}\} \ <\text{coproc}\>, \ {#}<\text{opc1}\>, \ <\text{Rt}\>, \ <\text{Rt2}\>, \ <\text{CRm}\>
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{coproc} & \text{CRm} & \text{opc1} \\
\hline
0b1111 & 0b1110 & 0b0011 \\
\hline
\end{array}
\]
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      UNDEFINED;
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
      CNTHVS_CVAL_EL2 = R[t2]:R[t];
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
      CNTHV_CVAL_EL2 = R[t2]:R[t];
    else
      CNTV_CVAL = R[t2]:R[t];
    endif
  endif
elsif PSTATE.EL == EL1 then
  CNTV_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
  CNTV_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
  CNTV_CVAL = R[t2]:R[t];
CNTHVS_TVAL, Counter-timer Secure Virtual Timer TimerValue Register (EL2)

The CNTHVS_TVAL characteristics are:

**Purpose**

Provides AArch32 access to the timer value for the Secure EL2 virtual timer.

**Note**

The Secure EL2 timer is implemented by ARMv8.4-SecEL2. It is only accessible from AArch32 state when EL2 is using AArch64 and the value of $\text{SCR}_\text{EL3}$.{$\text{EEL2}, \text{NS}$} is {1, 0}.

**Configuration**

AArch32 System register CNTHVS_TVAL bits [31:0] are architecturally mapped to AArch64 System register $\text{CNTHVS_TVAL}_\text{EL2}[31:0]$.

This register is present only when ARMv8.4-SecEL2 is implemented. Otherwise, direct accesses to CNTHVS_TVAL are UNDEFINED.

**Attributes**

CNTHVS_TVAL is a 32-bit register.

**Field descriptions**

The CNTHVS_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**TimerValue, bits [31:0]**

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If $\text{CNTHVS_CTL}_\text{ENABLE}$ is 0, the value returned is UNKNOWN.
- If $\text{CNTHVS_CTL}_\text{ENABLE}$ is 1, the value returned is ($\text{CNTHVS_CVAL}$ - $\text{CNTVCT}$).

On a write of this register, $\text{CNTHVS_CVAL}$ is set to ($\text{CNTVCT}$ + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When $\text{CNTHVS_CTL}_\text{ENABLE}$ is 1, the timer condition is met when ($\text{CNTVCT}$ - $\text{CNTHVS_CVAL}$) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- $\text{CNTHVS_CTL}_\text{ISTATUS}$ is set to 1.
- If $\text{CNTHVS_CTL}_\text{IMASK}$ is 0, an interrupt is generated.

When $\text{CNTHVS_CTL}_\text{ENABLE}$ is 0, the timer condition is not met, but $\text{CNTVCT}$ continues to count, so the TimerValue view appears to continue to count down.
Accessing the CNTHVS_TVAL

This register is accessed using the encoding for CNTV_TVAL.

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2}>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL;
    endif
elsif PSTATE.EL == EL1 then
    return CNTV_TVAL;
elsif PSTATE.EL == EL2 then
    return CNTV_TVAL;
elsif PSTATE.EL == EL3 then
    return CNTV_TVAL;
endif

\[
\text{MCR\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2}>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_TVAL_EL2 = R[t];
    else
        CNTV_TVAL = R[t];
    endif
elsif PSTATE.EL == EL1 then
    CNTV_TVAL = R[t];
elsif PSTATE.EL == EL2 then
    CNTV_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_TVAL = R[t];
endif
CNTKCTL, Counter-timer Kernel Control register

The CNTKCTL characteristics are:

**Purpose**

Controls the generation of an event stream from the virtual counter, and access from EL0 modes to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

**Configuration**

AArch32 System register CNTKCTL bits [31:0] are architecturally mapped to AArch64 System register CNTKCTL_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTKCTL is a 32-bit register.

**Field descriptions**

The CNTKCTL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | EVNTIS | RES0 | PLOPTEN | PLOVTEN | EVNTIE | EVNTDIR | EVNTEN | PLOVCTEN | PLOPCTEN |

**Bits [31:18]**

Reserved, RES0.

**EVNTIS, bit [17]**

When ARMv8.6-ECV is implemented:

Controls the scale of the generation of the event stream.

<table>
<thead>
<tr>
<th>EVNTIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CNTKCTL.EVNTI field applies to CNTVCT[15:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>The CNTKCTL.EVNTI field applies to CNTVCT[23:8].</td>
</tr>
</tbody>
</table>

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [16:10]**

Reserved, RES0.

**PLOPTEN, bit [9]**

Traps PL0 accesses to the physical timer registers to Undefined mode.
**PL0PTEN**

<table>
<thead>
<tr>
<th>PL0PTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PL0 accesses to the CNTP_CTL, CNTP_CVAL, and CNTP_TVAL registers are trapped to Undefined mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**PL0VTEN, bit [8]**

Traps PL0 accesses to the virtual timer registers to Undefined mode.

<table>
<thead>
<tr>
<th>PL0VTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PL0 accesses to the CNTV_CTL, CNTV_CVAL, and CNTV_TVAL registers are trapped to Undefined mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**EVNTI, bits [7:4]**

Selects which bit (0 to 15) of the counter register CNTVCT is the trigger for the event stream generated from that counter, when that stream is enabled.

This field resets to an architecturally **UNKNOWN** value.

**EVNTDIR, bit [3]**

Controls which transition of the counter register CNTVCT trigger bit, defined by EVNTI, generates an event when the event stream is enabled:

<table>
<thead>
<tr>
<th>EVNTDIR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A 0 to 1 transition of the trigger bit triggers an event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A 1 to 0 transition of the trigger bit triggers an event.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**EVNTEN, bit [2]**

Enables the generation of an event stream from the counter register CNTVCT.

<table>
<thead>
<tr>
<th>EVNTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables the event stream.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables the event stream.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**PL0VCTEN, bit [1]**

Traps PL0 accesses to the frequency register and virtual counter register to Undefined mode.

<table>
<thead>
<tr>
<th>PL0VCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PL0 accesses to the CNTVCT are trapped to Undefined mode. PL0 accesses to the CNTFRQ register are trapped to Undefined mode, if CNTKCTL.PLOPCTEN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**PL0PCTEN, bit [0]**

Traps PL0 accesses to the frequency register and physical counter register to Undefined mode.
PL0PCTEN | Meaning
---|---
0b0 | PL0 accesses to the CNTPCT are trapped to Undefined mode.
PL0 accesses to the CNTFRQ register are trapped to Undefined mode, if CNTKCTL.PL0VCTEN is also 0.
0b1 | This control does not cause any instructions to be trapped.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTKCTL**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} \langle coproc\rangle, \{#\}<opc1>, <Rt>, <CRn>, <CRm>\{, \{#\}<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL0 then UNDEFINED;}
\text{elsif PSTATE.EL == EL1 then return CNTKCTL;}
\text{elsif PSTATE.EL == EL2 then return CNTKCTL;}
\text{elsif PSTATE.EL == EL3 then return CNTKCTL;}
\]

\[
\text{MCR\{<c>\}{<q>} \langle coproc\rangle, \{#\}<opc1>, <Rt>, <CRn>, <CRm>\{, \{#\}<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL0 then UNDEFINED;}
\text{elsif PSTATE.EL == EL1 then CNTKCTL = R[t];}
\text{elsif PSTATE.EL == EL2 then CNTKCTL = R[t];}
\text{elsif PSTATE.EL == EL3 then CNTKCTL = R[t];}
\]
The CNTP_CTL characteristics are:

**Purpose**

Control register for the EL1 physical timer.

**Configuration**

AArch32 System register CNTP_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTP_CTL_EL0[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CNTP_CTL is a 32-bit register.

**Field descriptions**

The CNTP_CTL bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>ISTATUS</td>
<td>IMASK</td>
<td>ENABLE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is **UNKNOWN**.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:
Meaning

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally unknown value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTP_TVAL** continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to 0.

**Accessing the CNTP_CTL**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.E2H,TGE == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    end if;
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
    end if;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL0PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' then
    return CNTHPS_CTL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
    return CNTHP_CTL_EL2;
  else
    return CNTP_CTL;
  end if;
else if PSTATE_EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.NS == '0' then
    return CNTP_CTL_S;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.NS == '1' then
    return CNTP_CTL_NS;
  else
    return CNTP_CTL;
  end if;
else if PSTATE_EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return CNTP_CTL_NS;
  else
    return CNTP_CTL;
  end if;
else
  if SCR.NS == '0' then
    return CNTP_CTL_S;
  else
    return CNTP_CTL_NS;
  end if;
else
  return CNTP_CTL;
end if;
else
  if SCR.NS == '0' then
    return CNTP_CTL_S;
  else
    return CNTP_CTL_NS;
  end if;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHPS_CTL_EL2 = R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x03);
        elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                CNTP_CTL_S = R[t];
            else
                CNTP_CTL_NS = R[t];
        else
            CNTP_CTL = R[t];
        elseif PSTATE.EL == EL2 then
            if HaveEL(EL3) && ELUsingAArch32(EL3) then
                CNTP_CTL_NS = R[t];
            else
                CNTP_CTL = R[t];
        elseif PSTATE.EL == EL3 then
            if SCR.NS == '0' then
                CNTP_CTL_S = R[t];
            else
                CNTP_CTL_NS = R[t];
The CNTP_CVAL characteristics are:

**Purpose**

Holds the compare value for the EL1 physical timer.

**Configuration**

AArch32 System register CNTP_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTP_CVAL_EL0[63:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CNTP_CVAL is a 64-bit register.

**Field descriptions**

The CNTP_CVAL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**CompareValue**, bits [63:0]

Holds the EL1 physical timer CompareValue.

When CNTP_CTL ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- If CNTP_CTL.IMASK is 0, an interrupt is generated.

When CNTP_CTL ENABLE is 0, the timer condition is not met, but CNTPCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are **RES0**.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTP_CVAL**

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>}, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHTCTL_EL2.EL1PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHTCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHTCTL.PL1PTEN == '0' then
      AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHPS_CVAL_EL2;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHP_CVAL_EL2;
  else
    return CNTP_CVAL;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHTCTL_EL2.EL1PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CNTHTCTL_EL2.EL0PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHTCTL.PL1PTEN == '0' then
      AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
      return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
      return CNTHP_CVAL_EL2;
    else
      return CNTP_CVAL;
  elsif PSTATE.EL == EL2 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHTCTL_EL2.EL1PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CNTHTCTL_EL2.EL0PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHTCTL.PL1PTEN == '0' then
      AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
      return CNTHPS_CVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
      return CNTHP_CVAL_EL2;
    else
      return CNTP_CVAL;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return CNTP_CVAL_S;
    elseif SCR.NS == '1' then
      return CNTP_CVAL_NS;
    else
      return CNTP_CVAL;
  elsif PSTATE.EL == EL4 then
    if SCR.NS == '0' then
      return CNTP_CVAL_S;
    elseif SCR.NS == '1' then
      return CNTP_CVAL_NS;
    else
      return CNTP_CVAL;
else
  return CNTP_CVAL;

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    end if
elsif EL2Enabled() && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    end if
elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
else
    UNDEFINED;
end if
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch32.TakeHypTrapException(0x04);
else
    UNDEFINED;
end if
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    CNTHPS_CVAL_EL2 = R[t2]:R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    CNTHP_CVAL_EL2 = R[t2]:R[t];
else
    CNTP_CVAL = R[t2]:R[t];
end if
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CNTP_CVAL_S = R[t2]:R[t];
        else
            CNTP_CVAL_NS = R[t2]:R[t];
        end if
    else
        CNTP_CVAL = R[t2]:R[t];
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
    end if
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t2]:R[t];
    else
        CNTP_CVAL_NS = R[t2]:R[t];
    end if
The CNTP_TVAL characteristics are:

**Purpose**

Holds the timer value for the EL1 physical timer.

**Configuration**

AArch32 System register CNTP_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTP_TVAL_EL0[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTP_TVAL is a 32-bit register.

**Field descriptions**

The CNTP_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| TimerValue |    |

**TimerValue, bits [31:0]**

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL.ENABLE is 0, the value returned is **UNKNOWN**.
- If CNTP_CTL.ENABLE is 1, the value returned is (CNTP_CVAL - CNTPCT).

On a write of this register, CNTP_CVAL is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CNTP_CVAL) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- If CNTP_CTL.IMASK is 0, an interrupt is generated.

When CNTP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTP_TVAL**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        end
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    end
elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
        AArch32.TakeHypTrapException(0x00);
    else
        UNDEFINED;
    end
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHPS_TVAL_EL2;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHP_TVAL_EL2;
else
    return CNTP_TVAL;
end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return CNTP_TVAL_S;
        else
            return CNTP_TVAL_NS;
        end
    else
        return CNTP_TVAL;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_TVAL_NS;
    else
        return CNTP_TVAL_NS;
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_TVAL_S;
    else
        return CNTP_TVAL_NS;
    end
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.E2H,TGE) == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
    endif
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    CNTHPS_TVAL_EL2 = R[t];
  endif
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL0PTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
      AArch32.TakeHypTrapException(0x03);
    else
      CNTP_TVAL_S = R[t];
    endif
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      CNTP_TVAL_NS = R[t];
    else
      CNTP_TVAL = R[t];
    endif
  else
    if SCR.NS == '0' then
      CNTP_TVAL_NS = R[t];
    else
      CNTP_TVAL = R[t];
    endif
  endif
endif
CNTPCT, Counter-timer Physical Count register

The CNTPCT characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

AArch32 System register CNTPCT bits [63:0] are architecturally mapped to AArch64 System register CNTPCT_EL0[63:0].

All reads to the CNTPCT occur in program order relative to reads to CNTPCTSS or CNTPCT.

Attributes

CNTPCT is a 64-bit register.

Field descriptions

The CNTPCT bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Physical count value</td>
</tr>
<tr>
<td>62</td>
<td>Physical count value</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Physical count value</td>
</tr>
</tbody>
</table>

Bits [63:0]

Physical count value.

Accessing the CNTPCT

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && CNTKCTL_EL1.EL0PCTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  endif
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  endif
else
  AArch64.AArch32SystemAccessTrap(EL1, 0x04);
endif

if !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  UNDEFINED;
endif

if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHTCTL_EL2.EL1PCTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHTCTL_EL2.EL1PCTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
  AArch32.TakeHypTrapException(0x04);
else
  return CNTPCT;
endif

if !ELUsingAArch32(EL2) && CNTKCTL_EL2.EL0PCTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && !ELUsingAArch32(EL2) && CNTKCTL.PL1PCTEN == '0' then
  AArch32.TakeHypTrapException(0x04);
else
  return CNTPCT;
endif

if PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL2) && CNTKCTL_EL2.EL1PCTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elseif ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  endif
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  endif
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL0PCTEN == '0' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
else
  AArch64.AArch32SystemAccessTrap(EL2, 0x04);
endif

if EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
  AArch32.TakeHypTrapException(0x04);
else
  return CNTPCT;
endif

if PSTATE.EL == EL2 then
  return CNTPCT;
else
  return CNTPCT;
endif

if PSTATE.EL == EL3 then
  return CNTPCT;
else
  return CNTPCT;
endif

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
CNTPCTSS, Counter-timer Self-Synchronized Physical Count register

The CNTPCTSS characteristics are:

**Purpose**

Holds the 64-bit physical count value.

**Configuration**

AArch32 System register CNTPCTSS bits [63:0] are architecturally mapped to AArch64 System register CNTPCTSS_EL0[63:0].

This register is present only when ARMv8.6-ECV is implemented. Otherwise, direct accesses to CNTPCTSS are UNDEFINED.

All reads to the CNTPCTSS occur in program order relative to reads to CNTPCT or CNTPCTSS.

This register is a self-synchronised view of the CNTPCT counter, and cannot be read speculatively.

**Attributes**

CNTPCTSS is a 64-bit register.

**Field descriptions**

The CNTPCTSS bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |----|----|
| Self-Synchronized Physical count value |
| Self-Synchronized Physical count value |

**Bits [63:0]**

Self-Synchronized Physical count value.

**Accessing the CNTPCTSS**

Accesses to this register use the following encodings:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b1000</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') &&
        CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            endif
        endif
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' &&
        CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
        CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    else
        return CNTPCTSS;
    endif
else
    if PSTATE_EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
            CNTHCTL_EL2.EL0PCTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            return CNTPCTSS;
        endif
    elseif PSTATE_EL == EL2 then
        return CNTPCTSS;
    elseif PSTATE_EL == EL3 then
        return CNTPCTSS;
    else
        return CNTPCTSS;
    endif
endif
CNTV_CTL, Counter-timer Virtual Timer Control register

The CNTV_CTL characteristics are:

**Purpose**

Control register for the virtual timer.

**Configuration**

AArch32 System register CNTV_CTL bits [31:0] are architecturally mapped to AArch64 System register CNTV_CTL_EL0[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTV_CTL is a 32-bit register.

**Field descriptions**

The CNTV_CTL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ISTATUS | IMASK | ENABLE |

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:
<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>

For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from `CNTV_TVAL` continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to 0.

**Accessing the CNTV_CTL**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() & & !(ELUsingAArch32(EL2) && HCR_EL2.TGE == '1') then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        endif
    else
        if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        endif
    endif
elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
    if EL2Enabled() & & !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
        AArch32.TakeHypTrapException(0x00);
    else
        UNDEFINED;
    endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1TVT == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
    return CNTHVS_CTL_EL2;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
    return CNTHV_CTL_EL2;
else
    return CNTV_CTL;
elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return CNTV_CTL;
    endif
elsif PSTATE.EL == EL2 then
    return CNTV_CTL;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL;
end

MCR{<c>{<q>}} {coproc}, {#<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>proc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  endif
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
  CNTV_CTL = R[t];
elsif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  endif
elsif PSTATE.EL == EL2 then
  CNTV_CTL = R[t];
elsif PSTATE.EL == EL3 then
  CNTV_CTL = R[t];
CNTV_CVAL, Counter-timer Virtual Timer CompareValue register

The CNTV_CVAL characteristics are:

**Purpose**

Holds the compare value for the virtual timer.

**Configuration**

AArch32 System register CNTV_CVAL bits [63:0] are architecturally mapped to AArch64 System register CNTV_CVAL_EL0[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CNTV_CVAL is a 64-bit register.

**Field descriptions**

The CNTV_CVAL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>CompareValue</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**CompareValue, bits [63:0]**

Holds the EL1 virtual timer CompareValue.

When CNTV_CTL ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTV_CTL ISTATUS is set to 1.
- If CNTV_CTL IMASK is 0, an interrupt is generated.

When CNTV_CTL ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTV_CVAL**

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  end if
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  end if
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTKCTL_EL2.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTHCTL_EL2.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    return CNTV_CVAL;
  end if
elif PSTATE.EL == EL1 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL1VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    return CNTV_CVAL;
  end if
elsif PSTATE.EL == EL2 then
  return CNTV_CVAL;
elsif PSTATE.EL == EL3 then
  return CNTV_CVAL;

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  endif
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
    endif
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elseif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL0VTEN == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  endif
  elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '1') && CNTKCTL_EL1.EL1VTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    endif
  elsif PSTATE.EL == EL2 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VTEN == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    endif
  elseif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t2]:R[t];
  endif
else
  CNTV_CVAL = R[t2]:R[t];
endif
The CNTV_TVAL characteristics are:

**Purpose**

Holds the timer value for the virtual timer.

**Configuration**

AArch32 System register CNTV_TVAL bits [31:0] are architecturally mapped to AArch64 System register CNTV_TVAL_EL0[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

CNTV_TVAL is a 32-bit register.

**Field descriptions**

The CNTV_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**TimerValue, bits [31:0]**

The TimerValue view of the virtual timer.

On a read of this register:

- If **CNTV_CTL.ENABLE** is 0, the value returned is **UNKNOWN**.
- If **CNTV_CTL.ENABLE** is 1, the value returned is (**CNTV_CVAL** - **CNTVCT**).

On a write of this register, **CNTV_CVAL** is set to (**CNTVCT** + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When **CNTV_CTL.ENABLE** is 1, the timer condition is met when (**CNTVCT** - **CNTP_CVAL**) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- **CNTV_CTL.ISTATUS** is set to 1.
- If **CNTV_CTL.IMASK** is 0, an interrupt is generated.

When **CNTV_CTL.ENABLE** is 0, the timer condition is not met, but **CNTVCT** continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTV_TVAL**

Accesses to this register use the following encodings:

$$\text{MRC\{<c>\}{<q>}} \ <\text{coproc}, \ \{\#<opc1>, \ \langle\text{Rt}\rangle, \ \langle\text{CRn}\rangle, \ \langle\text{CRm}\rangle, \ \{\#<opc2>\}}$$
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VT
    == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL;
    end
elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VT
    == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        return CNTHVS_TVAL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_TVAL_EL2;
    else
        return CNTV_TVAL;
    end
elsif PSTATE.EL == EL2 then
    return CNTV_TVAL;
elsif PSTATE.EL == EL3 then
    return CNTV_TVAL;
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VT
    == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_TVAL_EL2 = R[t];
    else
        CNTV_TVAL = R[t];
    end
elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTHCTL_EL2.EL1VT
    == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' then
        CNTHVS_TVAL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_TVAL_EL2 = R[t];
    else
        CNTV_TVAL = R[t];
    end
elsif PSTATE.EL == EL2 then
    CNTV_TVAL = R[t];
elsif PSTATE.EL == EL3 then
    CNTV_TVAL = R[t];
CNTVCT, Counter-timer Virtual Count register

The CNTVCT characteristics are:

**Purpose**

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value minus the virtual offset visible in CNTVOFF.

**Configuration**

AArch32 System register CNTVCT bits [63:0] are architecturally mapped to AArch64 System register CNTVCT_EL0[63:0].

The value of this register is the same as the value of CNTPCT in the following conditions:

- When EL2 is not implemented.
- When EL2 is implemented and is using AArch64, HCR_EL2.E2H, TGE is {1, 1}, and this register is read from Non-secure EL0.

All reads to the CNTVCT occur in program order relative to reads to CNTVCTSS or CNTVCT.

**Attributes**

CNTVCT is a 64-bit register.

**Field descriptions**

The CNTVCT bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:0]</th>
<th>Virtual count value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
<td>Virtual count value</td>
</tr>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>Virtual count value</td>
</tr>
</tbody>
</table>

**Accessing the CNTVCT**

Accesses to this register use the following encodings:

\[
\text{MRRC}\{<c>\}{<q>} \text{<coproc>, } \{#\}<\text{opc1}>, <\text{Rt}>, <\text{Rt2}>, <\text{CRm}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTKCTL_EL1.EL0VCTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VCTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
  CNTHCTL_EL2.EL0VCT == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTHCTL_EL2.EL1TVCT == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    return CNTVCT;
  elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTHCTL_EL2.EL1TVCT == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      return CNTVCT;
  elsif PSTATE.EL == EL2 then
    return CNTVCT;
  elsif PSTATE.EL == EL3 then
    return CNTVCT;
CNTVCTSS, Counter-timer Self-Synchronized Virtual Count register

The CNTVCTSS characteristics are:

**Purpose**

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value visible in CNTPCT minus the virtual offset visible in CNTVOFF.

**Configuration**

AArch32 System register CNTVCTSS bits [63:0] are architecturally mapped to AArch64 System register CNTVCTSS_EL0[63:0].

This register is present only when ARMv8.6-ECV is implemented. Otherwise, direct accesses to CNTVCTSS are UNDEFINED.

All reads to the CNTVCTSS occur in program order relative to reads to CNTPCT or CNTVCTSS.

This register is a self-synchronised view of the CNTPCT counter, and cannot be read speculatively.

**Attributes**

CNTVCTSS is a 64-bit register.

**Field descriptions**

The CNTVCTSS bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Self-Synchronized Virtual count value</td>
</tr>
<tr>
<td>62</td>
<td>Self-Synchronized Virtual count value</td>
</tr>
<tr>
<td>31</td>
<td>Bits [63:0]</td>
</tr>
<tr>
<td>0</td>
<td>Self-Synchronized Virtual count value</td>
</tr>
</tbody>
</table>

Accessing the CNTVCTSS

Accesses to this register use the following encodings:

```
MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b1001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTKCTL_EL1.EL0VCTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x04);
  elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VCTEN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
  CNTHCTL_EL2.EL0VCT == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
  CNTHCTL_EL2.EL1TVCT == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  else
    return CNTVCTSS;
  elsif PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
    CNTHCTL_EL2.EL1VCTEN == '0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
      UNDEFINED;
  elsif PSTATE.EL == EL2 then
    return CNTVCTSS;
  elsif PSTATE.EL == EL3 then
    return CNTVCTSS;
CNTVOFF, Counter-timer Virtual Offset register

The CNTVOFF characteristics are:

**Purpose**

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in CNTPCT and the virtual count value visible in CNTVCT.

**Configuration**

AArch32 System register CNTVOFF bits [63:0] are architecturally mapped to AArch64 System register CNTVOFF_EL2[63:0].

If EL2 is not implemented, this register is RES0 from EL3 and the virtual counter uses a fixed virtual offset of zero.

**Note**

When EL2 is implemented and is using AArch64, if HCR_EL2.{E2H, TGE} is \{1, 1\}, the virtual counter uses a fixed virtual offset of zero when CNTVCT is read from Non-secure EL0.

When EL2 is implemented and can use AArch32, on a reset into an Exception level that is using AArch32 this register resets to an IMPLEMENTATION DEFINED value that might be UNKNOWN.

**Attributes**

CNTVOFF is a 64-bit register.

**Field descriptions**

The CNTVOFF bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Virtual offset**

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTVOFF**

Accesses to this register use the following encodings:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTVOFF;
elsif PSTATE.EL == EL3 then
    return CNTVOFF;

MCRR{<c>}{<q>}<coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>
CONTEXTIDR, Context ID Register

The CONTEXTIDR characteristics are:

**Purpose**

Identifies the current Process Identifier and, when using the Short-descriptor translation table format, the Address Space Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

**Configuration**

AArch32 System register CONTEXTIDR bits [31:0] are architecturally mapped to AArch64 System register CONTEXTIDR_EL1[31:0].

The register format depends on whether address translation is using the Long-descriptor or the Short-descriptor translation table format.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CONTEXTIDR is a 32-bit register.

**Field descriptions**

The CONTEXTIDR bit assignments are:

**When TTBCR.EAE == 0:**

![Field assignments for TTBCR.EAE == 0]

PROCID, bits [31:8]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

This field resets to an architecturally UNKNOWN value.

ASID, bits [7:0]

Address Space Identifier. This field is programmed with the value of the current ASID.

This field resets to an architecturally UNKNOWN value.

**When TTBCR.EAE == 1:**

![Field assignments for TTBCR.EAE == 1]
**PROCID, bits [31:0]**

Process Identifier. This field must be programmed with a unique value that identifies the current process.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CONTEXTIDR**

Accesses to this register use the following encodings:

\[
\text{MRC} \{<c>\} \{<q>\} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#<\text{opc2}>\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return CONTEXTIDR_S;
    else
      return CONTEXTIDR_NS;
  else
    return CONTEXTIDR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAAArch32(EL3) then
    return CONTEXTIDR_NS;
  else
    return CONTEXTIDR;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return CONTEXTIDR_S;
  else
    return CONTEXTIDR_NS;
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      CONTEXTIDR_S = R[t];
    else
      CONTEXTIDR_NS = R[t];
    end if;
  else
    CONTEXTIDR = R[t];
  end if;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    CONTEXTIDR_NS = R[t];
  else
    CONTEXTIDR = R[t];
  end if;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    CONTEXTIDR_S = R[t];
  else
    CONTEXTIDR_NS = R[t];
  end if;
CP15DMB, Data Memory Barrier System instruction

The CP15DMB characteristics are:

**Purpose**

Performs a Data Memory Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the DMB instruction instead.

**Configuration**

**Attributes**

CP15DMB is a 32-bit System instruction.

**Field descriptions**

CP15DMB ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the CP15DMB instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}\{<q>\} \ <\text{coproc}>,\ {#}<\text{opc1}>,\ <\text{Rt}>,\ <\text{CRn}>,\ <\text{CRm}\{,\ {#}<\text{opc2}}>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN == '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DMB();
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
            UNDEFINED;
        else
            CP15DMB();
        elsif PSTATE.EL == EL2 then
            if HSCTRL.CP15BEN == '0' then
                UNDEFINED;
            else
                CP15DMB();
            elsif PSTATE.EL == EL3 then
                CP15DMB();
            end if
        end if
    end if
end

CP15DSB, Data Synchronization Barrier System instruction

The CP15DSB characteristics are:

**Purpose**

Performs a Data Synchronization Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the DSB instruction instead.

**Configuration**

**Attributes**

CP15DSB is a 32-bit System instruction.

**Field descriptions**

CP15DSB ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the CP15DSB instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR\{<c>\}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN == '0' then
    UNDEFINED;
  elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    CP15DSB();
  endif
else if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    CP15DSB();
  endif
else if PSTATE.EL == EL2 then
  if HSCTRL.CP15BEN == '0' then
    UNDEFINED;
  else
    CP15DSB();
  endif
else if PSTATE.EL == EL3 then
  CP15DSB();
end if
CP15ISB, Instruction Synchronization Barrier System instruction

The CP15ISB characteristics are:

**Purpose**

Performs an Instruction Synchronization Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the ISB instruction instead.

**Configuration**

**Attributes**

CP15ISB is a 32-bit System instruction.

**Field descriptions**

CP15ISB ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the CP15ISB instruction**

Accesses to this instruction use the following encodings:

\[ \text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}> \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN == '0' then
    UNDEFINED;
  elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    CP15ISB();
  elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
      UNDEFINED;
    else
      CP15ISB();
  elseif PSTATE.EL == EL2 then
    if HSCTRL.CP15BEN == '0' then
      UNDEFINED;
    else
      CP15ISB();
  elseif PSTATE.EL == EL3 then
    CP15ISB();

CPACR, Architectural Feature Access Control Register

The CPACR characteristics are:

**Purpose**

Controls access to trace, and to Advanced SIMD and floating-point functionality from EL0, EL1, and EL3.

In an implementation that includes EL2, the CPACR has no effect on instructions executed at EL2.

**Configuration**

AArch32 System register CPACR bits [31:0] are architecturally mapped to AArch64 System register CPACR_EL1[31:0].

Bits in the NSACR control Non-secure access to the CPACR fields. See the field descriptions for more information.

**Note**

In the register field descriptions, controls are described as applying at specified Privilege levels. This is because, in Secure state, a PL1 control:

- Applies to execution in a Secure EL3 mode when EL3 is using AArch32.
- Applies to execution in a Secure EL1 mode when EL3 is using AArch64.

See ‘Security state, Exception levels, and AArch32 execution privilege’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.7.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CPACR is a 32-bit register.

**Field descriptions**

The CPACR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ASE | DIS | RES | 0 | TR | CD | DIS | RES | 0 | cp11 | cp10 | RES | 0 |

**ASEDIS, bit [31]**

Disables PL0 and PL1 execution of Advanced SIMD instructions.

<table>
<thead>
<tr>
<th>ASE</th>
<th>DIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control permits execution of Advanced SIMD instructions at PL0 and PL1.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>All instruction encodings that are Advanced SIMD instruction encodings, but are not also floating-point instruction encodings, are UNDEFINED at PL0 and PL1.</td>
<td></td>
</tr>
</tbody>
</table>

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, it is RAZ/WI.
If EL3 is implemented and is using AArch32, and the value of NSACR.NSASEDIS is 1, this field behaves as RAO/WI in Non-secure state, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see ‘Controls of Advanced SIMD operation that do not apply to floating-point operation’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.

See the description of CPACR.cp10 for a list of other controls that can disable or trap execution of Advanced SIMD instructions in AArch32 state.

This field resets to 0.

**Bits [30:29]**

Reserved, RES0.

**TRCDIS, bit [28]**

Traps PL0 and PL1 System register accesses to all implemented trace registers to Undefined mode.

<table>
<thead>
<tr>
<th>TRCDIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on PL0 and PL1 System register accesses to trace registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>PL0 and PL1 System register accesses to all implemented trace registers are trapped to Undefined mode.</td>
</tr>
</tbody>
</table>

If the implementation does not include a PE trace unit, or does not include a System register interface to the PE trace unit registers, this field is RES0. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSSTRCDIS is 1, this field behaves as RAO/WI in Non-secure state, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

**Note**

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the implementation includes an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED.
- The architecture does not provide traps on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

This field resets to an architecturally UNKNOWN value.

**Bits [27:24]**

Reserved, RES0.

**cp11, bits [23:22]**

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the CPACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In Non-secure state, if EL3 is implemented and is using AArch32, when the value of NSACR.cp10 is 0, this field behaves as RAZ/WI, regardless of its actual value.

This field resets to 0.

**cp10, bits [21:20]**

Defines the access rights for the floating-point and Advanced SIMD functionality. Possible values of the field are:
The floating-point and Advanced SIMD features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

### Note

The CPACR has no effect on floating-point and Advanced SIMD accesses from PL2. These can be disabled by the HCPTR.TCP10 field.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In Non-secure state, if EL3 is implemented and is using AArch32, when the value of NSACR.cp10 is 0, this field behaves as RAZ/WI, regardless of its actual value.

Execution of floating-point and Advanced SIMD instructions in AArch32 state can be disabled or trapped by the following controls:

- CPACR.cp10, or, if executing at EL0, CPACR_EL1.FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
  - HCPTR.TCP10, or if EL2 is using AArch64, CPTR_EL2.TFP.
  - NSACR.cp10, or if EL3 is using AArch64, CPTR_EL3.TFP.
- For Advanced SIMD instructions only:
  - CPACR.ASEDIS.
  - If executing in Non-secure state, HCPTR.TASE and NSACR.NSTRCDIS.

See the descriptions of the controls for more information.

This field resets to 0.

### Bits [19:0]

Reserved, RES0.

### Accessing the CPACR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TCPAC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return CPACR;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      CPACR = R[t];
  elsif PSTATE.EL == EL3 then
    return CPACR;

MCR\{coproc\}\{opc1\} \{CRn\} \{CRm\} \{opc2\}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TCPAC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    CPACR = R[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      CPACR = R[t];
  elsif PSTATE.EL == EL3 then
    CPACR = R[t];
The CPPRCTX characteristics are:

**Purpose**

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, cache prefetch prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

**Note**

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

**Configuration**

This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to CPPRCTX are UNDEFINED.

**Attributes**

CPPRCTX is a 32-bit System instruction.

**Field descriptions**

The CPPRCTX input value bit assignments are:

```
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | GVMID | NS | EL | VMID | RES0 | GASID | ASID |
```

**Bits [31:28]**

Reserved, RES0.

**GVMID, bit [27]**

Execution of this instruction applies to all VMIDs or a specified VMID.
GVMID

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

NS, bit [26]

Security State.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state.</td>
</tr>
</tbody>
</table>

If the instruction is executed in Non-secure state, this field is treated as 1.

EL, bits [25:24]

Exception Level.

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>EL0.</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>EL3.</td>
</tr>
</tbody>
</table>

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and either:

- an EL1 context.
- an EL0 context when \( \text{HCR_EL2.E2H} == 0 \) or \( \text{HCR_EL2.TGE} == 0 \) or EL2 is using AArch32 state.

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and \( \text{HCR_EL2.E2H} == 0 \) or \( \text{HCR_EL2.TGE} == 0 \) or \( \text{ELUsingAArch32(EL2)} \) then this field is treated as the current VMID.

When the instruction is executed at EL0 and \( \text{HCR_EL2.E2H} == 1 \) and \( \text{HCR_EL2.TGE} == 1 \) and \( \text{ELUsingAArch32(EL2)} \) then this field is ignored.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

<table>
<thead>
<tr>
<th>GASID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0, then this field has an Effective value of 0.
ASID, bits [7:0]

Only applies for an EL0 context and when bit[8] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.

Executing the CPPRCTX instruction

Accesses to this instruction use the following encodings:

\[
\text{MCR}{\langle c \rangle}{\langle q \rangle} \langle \text{coproc} \rangle, \{\#\langle \text{opcl} \rangle, \langle \text{Rt} \rangle, \langle \text{CRn} \rangle, \langle \text{CRm} \rangle, \{\#\langle \text{opc2} \rangle \}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !EL2Enabled() && HCR_EL2.<E2H,TGE> == '1' &&&& SCTLR_EL1.EnRCTX == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  endif
endif

elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
else
  if EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x03);
  else
    CPPRCTX(R[t]);
  endif
endif

elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x03);
  else
    CPPRCTX(R[t]);
  endif
elsif PSTATE.EL == EL2 then
  CPPRCTX(R[t]);
elsif PSTATE.EL == EL3 then
  CPPRCTX(R[t]);

CPSR, Current Program Status Register

The CPSR characteristics are:

**Purpose**

Holds PE status and control information.

**Configuration**

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CPSR is a 32-bit register.

**Field descriptions**

The CPSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| N  | Z  | C  | V  | Q  | RES0 | SSBS | PAN | DIT | RES0 | GE | RES0 | E | A | I | F | RES0 | RES1 | M |

**N, bit [31]**

Negative condition flag. Set to bit[31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

**Z, bit [30]**

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

**C, bit [29]**

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

**V, bit [28]**

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

**Q, bit [27]**

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

**Bits [26:24]**

Reserved, RES0.

**SSBS, bit [23]**
When ARMv8.0-SSBS is implemented:

Speculative Store Bypass Safe.

Prohibits speculative loads or stores which might practically allow a cache timing side channel.

A cache timing side channel might be exploited where a load or store uses an address that is derived from a register that is being loaded from memory using a load instruction speculatively read from a memory location. If PSTATE.SSBS is enabled, the address derived from the load instruction might be from earlier in the coherence order than the latest store to that memory location with the same virtual address.

<table>
<thead>
<tr>
<th>SSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hardware is not permitted to load or store speculatively in the manner described.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hardware is permitted to load or store speculatively in the manner described.</td>
</tr>
</tbody>
</table>

The value of this bit is usually set to the value described by the SCTLR.DSSBS bit on exceptions to any mode except Hyp mode, and the value described by HSCTLR.DSSBS on exceptions to Hyp mode.

This field resets to an implementation defined value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never.

<table>
<thead>
<tr>
<th>PAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation system is the same as Armv8.0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Disables privileged read and write accesses to addresses accessible at EL0.</td>
</tr>
</tbody>
</table>

The value of this bit is usually preserved on taking an exception, except in the following situations:

- When the target of the exception is EL1, and the value of the SCTLR.SPAN bit for the current Security state is 0, this bit is set to 1.
- When the target of the exception is EL3, from Secure state, and the value of the Secure SCTLR.SPAN is 0, this bit is set to 1.
- When the target of the exception is EL3, from Non-secure state, this bit is set to 0 regardless of the value of the Secure SCTLR.SPAN bit.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When ARMv8.4-DIT is implemented:

Data Independent Timing.
The architecture makes no statement about the timing properties of any instructions.

The architecture requires that:

- The timing of every load and store instruction is insensitive to the value of the data being loaded or stored.
- For certain data processing instructions, the instruction takes a time which is independent of:
  - The values of the data supplied in any of its registers.
  - The values of the NZCV flags.
- For certain data processing instructions, the response of the instruction to asynchronous exceptions does not vary based on:
  - The values of the data supplied in any of its registers.
  - The values of the NZCV flags.

The data processing instructions affected by this bit are:

- All cryptographic instructions. These instructions are:
  - AESD, ASE, AESIMC, AESMC, SHA1C, SHA1H, SHA1M, SHA1P, SHA1SU0, SHA1SU1, SHA256H, SHA256H2, SHA256SU0, and SHA256SU1.

- A subset of those instructions which use the general-purpose register file. For these instructions, the effects of CPSR.DIT apply only if they do not use R15 as either their source or destination and pass their condition execution check. The instructions are:

- A subset of those instructions which use the general-purpose register file. For these instructions, the effects of CPSR.DIT do not depend on these instructions passing their condition execution check. These instructions are:
  - ADC (immediate), ADC (register), ADCA (register), ADCS (register), ADD (immediate), ADD (register), ADDS (immediate), ADDS (register), AND (immediate), AND (register), ANDS (immediate), ANDS (register), ASR (immediate), ASR (register), ASRS (immediate), ASRS (register), BIC (immediate), BIC (register), BICS (immediate), BICS (register), EOR (immediate), EOR (register), EORS (register), EORS (register), LSL (immediate), LSL (register), LSLS (immediate), LSLS (register), LSR (immediate), LSR (register), LSRS (immediate), LSRS (register), MOV (immediate), MOV (register), MOV(S) (immediate), MOV(S) (register), MVNS (immediate), MVNS (register), MVNS (immediate), MVNS (register), ORR (immediate), ORR (register), ORRS (immediate), ORRS (register), ROR (immediate), ROR (register), RORS (immediate), RORS (register), RSB (immediate), RSB (register), RSB(S) (immediate), RSB(S) (register), RSC (immediate), RSC (register), RSC(S) (immediate), RSC(S) (register), SUB (immediate), SUB (register), SUBS (immediate), and SUBS (register).

- A subset of those instructions which use the SIMD&FP register file. For these instructions, the effects of CPSR.DIT apply only if they pass their condition execution check. These instructions are:
  - CRC32B, CRC32H, CRC32W, CRC32WB, CRC32CH, CRC32CW, VABA*, VABD*, VABS, VACGE, VACGT, VACLE, VACL, VADD (integer), VADDH, VADDL, VADDV, VAND, VBIC, VBIT, VBSL, VCGE, VCGT, VCLE, VCLS, VCLT, VCLZ, VCMP, VCMPX, VCNT, VDUP, VEOR, VEXT, VHADD, VHSUB, VMAX (integer), VMIN (integer), VMR (integer), VMRL (integer), VMRAL, VMLS (integer), VMLS, VMOV, VMOVQ, VMOVN, VMUL (integer and polynomial), VMULL (integer and polynomial), VMVN, VNEG, VORN, VORR, VPADAL, VPADD (integer), VPADDL, VPMAX (integer),
VPMIN (integer), VRADDHN, VREV*, VRHADD, VRSHL, VRSHR, VSHLHN, VRSRA, VRSUBHN, VSELEQ, VSELGE, VSELGT, VSELVS, VSHL, VSHLHN, VSHR, VSLI, VSLRA, VSRI, VSUB (integer), VSUBHN, VSUBL, VSWP, VTBX, VTBN, VTST, VUZP, and VZIP

This field resets to 0.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

Bits [15:10]

Reserved, RES0.

E, bit [9]

Endianness state bit. Controls the load and store endianness for data accesses:

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Little-endian operation</td>
</tr>
<tr>
<td>0b1</td>
<td>Big-endian operation.</td>
</tr>
</tbody>
</table>

Instruction fetches ignore this bit.

If an implementation does not provide Big-endian support, this bit is RES0. If it does not provide Little-endian support, this bit is RES1.

If an implementation provides Big-endian support but only at EL0, this bit is RES0 for an exception return to any Exception level other than EL0.

Likewise, if it provides Little-endian support only at EL0, this bit is RES1 for an exception return to any Exception level other than EL0.

When the reset value of the SCTLR.EE bit is defined by a configuration input signal, that value also applies to the CPSR.E bit on reset, and therefore applies to software execution from reset.

A, bit [8]

SError interrupt mask bit. The possible values of this bit are:

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked.</td>
</tr>
</tbody>
</table>

I, bit [7]

IRQ mask bit. The possible values of this bit are:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked.</td>
</tr>
</tbody>
</table>
F, bit [6]

FIQ mask bit. The possible values of this bit are:

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked.</td>
</tr>
</tbody>
</table>

Bit [5]

Reserved, RES0.

Bit [4]

Reserved, RES1.

M, bits [3:0]

Current PE mode. Possible values are:

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>User.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Monitor.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b1111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Accessing the CPSR

CPSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions. For more details, see MRS, MSR (register), and MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
CSSEL R, Cache Size Selection Register

The CSSEL R characteristics are:

**Purpose**

Selects the current Cache Size ID Register, CSIDR, by specifying the required cache level and the cache type, which is either instruction cache or data cache.

If ARMv8.3-CCIDX is implemented, CSSEL R also selects the current CSIDR 2.

**Configuration**

AArch32 System register CSSEL R bits [31:0] are architecturally mapped to AArch64 System register CSSEL R_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

CSSEL R is a 32-bit register.

**Field descriptions**

The CSSEL R bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 | Level | InD |

**Bits [31:4]**

Reserved, RES0.

**Level, bits [3:1]**

Cache level of required cache. Permitted values are:

<table>
<thead>
<tr>
<th>Level</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Level 1 cache.</td>
</tr>
<tr>
<td>0b001</td>
<td>Level 2 cache.</td>
</tr>
<tr>
<td>0b010</td>
<td>Level 3 cache.</td>
</tr>
<tr>
<td>0b011</td>
<td>Level 4 cache.</td>
</tr>
<tr>
<td>0b100</td>
<td>Level 5 cache.</td>
</tr>
<tr>
<td>0b101</td>
<td>Level 6 cache.</td>
</tr>
<tr>
<td>0b110</td>
<td>Level 7 cache.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If CSSEL R.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSEL R is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**InD, bit [0]**

Instruction not Data bit. Permitted values are:
<table>
<thead>
<tr>
<th>InD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Data or unified cache.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction cache.</td>
</tr>
</tbody>
</table>

If CSSELR.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is **UNKNOWN**.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the CSSELR

Accesses to this register use the following encodings:

\[
\text{MRC\{}<c>\}\{}<q>\} \text{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>\}, \{#<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b010</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & ELUsingAAArch32(EL2) & HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & ELUsingAAArch32(EL2) & HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() & !ELUsingAAArch32(EL2) & HCR_EL2.TID2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & !ELUsingAAArch32(EL2) & HCR_EL2.TID4 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & !ELUsingAAArch32(EL2) & HCR.TID2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() & !ELUsingAAArch32(EL2) & HCR2.TID4 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) & ELUsingAAArch32(EL3) then
    if SCR.NS == '0' then
      return CSSELR_S;
    else
      return CSSELR_NS;
    end;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) & ELUsingAAArch32(EL3) then
      return CSSELR_NS;
    else
      return CSSELR;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return CSSELR_S;
    else
      return CSSELR_NS;
  end;

\[
\text{MCR\{}<c>\}\{}<q>\} \text{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>\}, \{#<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b010</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            CSSELR_S = R[t];
        else
            CSSELR_NS = R[t];
        end
    else
        CSSELR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CSSELR_NS = R[t];
    else
        CSSELR = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CSSELR_S = R[t];
    else
        CSSELR_NS = R[t];
    end
CTR, Cache Type Register

The CTR characteristics are:

**Purpose**

Provides information about the architecture of the caches.

**Configuration**

AArch32 System register CTR bits [31:0] are architecturally mapped to AArch64 System register `CTR_EL0[31:0]`.

**Attributes**

CTR is a 32-bit register.

**Field descriptions**

The CTR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>DIC</td>
<td>Instruction cache invalidation requirements for instruction to data coherence. The meaning of this bit is:</td>
</tr>
<tr>
<td>28</td>
<td>IDC</td>
<td>Data cache clean requirements for instruction to data coherence. The meaning of this bit is:</td>
</tr>
<tr>
<td>27</td>
<td>CWG</td>
<td>Cache writeback granule. Log2 of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified.</td>
</tr>
</tbody>
</table>

### Bit [31]

Reserved, RES1.

### Bit [30]

Reserved, RES0.

### DIC, bit [29]

Instruction cache invalidation requirements for instruction to data coherence. The meaning of this bit is:

<table>
<thead>
<tr>
<th>DIC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction cache invalidation to the Point of Unification is required for instruction to data coherence.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction cache invalidation to the Point of Unification is not required for instruction to data coherence.</td>
</tr>
</tbody>
</table>

### IDC, bit [28]

Data cache clean requirements for instruction to data coherence. The meaning of this bit is:

<table>
<thead>
<tr>
<th>IDC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Data cache clean to the Point of Unification is required for instruction to data coherence, unless CLIDR.LoC == 0b000 or (CLIDR.LoUIS == 0b000 &amp;&amp; CLIDR.LoUU == 0b000).</td>
</tr>
<tr>
<td>0b1</td>
<td>Data cache clean to the Point of Unification is not required for instruction to data coherence.</td>
</tr>
</tbody>
</table>
CTR, Cache Type Register

A value of 0b0000 indicates that this register does not provide Cache writeback granule information and either:

- The architectural maximum of 512 words (2KB) must be assumed.
- The Cache writeback granule can be determined from maximum cache line size encoded in the Cache Size ID Registers.

Values greater than 0b1001 are reserved.

Arm recommends that an implementation that does not support cache write-back implements this field as 0b0001. This applies, for example, to an implementation that supports only write-through caches.

**ERG, bits [23:20]**

Exclusives reservation granule. \( \log_2 \) of the number of words of the maximum size of the reservation granule that has been implemented for the Load-Exclusive and Store-Exclusive instructions.

The use of the value 0b0000 is deprecated.

The value 0b0001 and values greater than 0b1001 are reserved.

**DminLine, bits [19:16]**

\( \log_2 \) of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE.

**L1lp, bits [15:14]**

Level 1 instruction cache policy. Indicates the indexing and tagging policy for the L1 instruction cache. Possible values of this field are:

<table>
<thead>
<tr>
<th>L1lp</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>VMID aware Physical Index, Physical tag (VPIPT)</td>
</tr>
<tr>
<td>0b01</td>
<td>ASID-tagged Virtual Index, Virtual Tag (AIVVT)</td>
</tr>
<tr>
<td>0b10</td>
<td>Virtual Index, Physical Tag (VIPT)</td>
</tr>
<tr>
<td>0b11</td>
<td>Physical Index, Physical Tag (PIPT)</td>
</tr>
</tbody>
</table>

The value 0b01 is reserved in Armv8.

The value 0b00 is permitted only in an implementation that includes ARMv8.2-PIPTV, otherwise the value is reserved.

**Bits [13:4]**

Reserved, RES0.

**IminLine, bits [3:0]**

\( \log_2 \) of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE.

**Accessing the CTR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \text{ <coproc>, } \{#<opc1>, <Rt>, <CRn>, <CRm>\}, \{#\}<opc2>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return CTR;
  endif
elsif PSTATE.EL == EL2 then
  return CTR;
elsif PSTATE.EL == EL3 then
  return CTR;
DACR, Domain Access Control Register

The DACR characteristics are:

**Purpose**

 Defines the access permission for each of the sixteen memory domains.

**Configuration**

AArch32 System register DACR bits [31:0] are architecturally mapped to AArch64 System register DACR32_EL2[31:0].

This register has no function when TTBCR.EAE is set to 1, to select the Long-descriptor translation table format.

RW fields in this register reset to architecturally \textit{UNKNOWN} values.

**Attributes**

DACR is a 32-bit register.

**Field descriptions**

The DACR bit assignments are:

\[
\begin{array}{cccccccccccccccc}
\text{D15} & \text{D14} & \text{D13} & \text{D12} & \text{D11} & \text{D10} & \text{D9} & \text{D8} & \text{D7} & \text{D6} & \text{D5} & \text{D4} & \text{D3} & \text{D2} & \text{D1} & \text{D0} \\
\end{array}
\]

\textbf{D<n>}, bits [2n+1:2n], for n = 0 to 15

Domain n access permission, where n = 0 to 15. Permitted values are:

<table>
<thead>
<tr>
<th>D&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>No access. Any access to the domain generates a Domain fault.</td>
</tr>
<tr>
<td>0b01</td>
<td>Client. Accesses are checked against the permission bits in the translation tables.</td>
</tr>
<tr>
<td>0b11</td>
<td>Manager. Accesses are not checked against the permission bits in the translation tables.</td>
</tr>
</tbody>
</table>

The value 0b10 is reserved.

This field resets to an architecturally \textit{UNKNOWN} value.

**Accessing the DACR**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q}> } \text{<coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>, \{#\}<opc2>}
\]

\[
\begin{array}{|c|c|c|c|}
\hline
\text{coproc} & \text{opc1} & \text{CRn} & \text{CRm} \\
\hline
0b1111 & 0b000 & 0b0011 & 0b0000 \\
\hline
\end{array}
\]
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return DACR_S;
    else
      return DACR_NS;
  else
    return DACR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return DACR_NS;
  else
    return DACR;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return DACR_S;
  else
    return DACR_NS;

MCR{<c}>{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            DACR_S = R[t];
        else
            DACR_NS = R[t];
        end
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            DACR_NS = R[t];
        else
            DACR = R[t];
        end
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' && CP15SDISABLE == HIGH then
            UNDEFINED;
        elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                DACR_S = R[t];
            else
                DACR_NS = R[t];
            end
        end
    else
        DACR = R[t];
    end
end

DACR, Domain Access Control Register

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGAUTHSTATUS, Debug Authentication Status register

The DBGAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

AArch32 System register DBGAUTHSTATUS bits [31:0] are architecturally mapped to AArch64 System register DBGAUTHSTATUS_EL1[31:0].

AArch32 System register DBGAUTHSTATUS bits [31:0] are architecturally mapped to External register DBGAUTHSTATUS_EL1[31:0].

This register is required in all implementations.

Attributes

DBGAUTHSTATUS is a 32-bit register.

Field descriptions

The DBGAUTHSTATUS bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

SNID, bits [7:6]

When ARMv8.4-Debug is implemented:

Secure Non-Invasive Debug.

This field has the same value as DBGAUTHSTATUS.SID.

Otherwise:

Secure Non-Invasive Debug.

<table>
<thead>
<tr>
<th>SNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_NS is 1.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.
**SID, bits [5:4]**

Secure Invasive Debug.

<table>
<thead>
<tr>
<th>SID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0b00</code></td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1.</td>
</tr>
<tr>
<td><code>0b10</code></td>
<td>Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td><code>0b11</code></td>
<td>Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NSNID, bits [3:2]**

When ARMv8.4-Debug is implemented:

Non-secure Non-invasive debug.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0b00</code></td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR.NS is 0.</td>
</tr>
<tr>
<td><code>0b11</code></td>
<td>Implemented and enabled. EL3 is implemented or the Effective value of SCR.NS is 1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Non-secure Non-Invasive Debug.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0b00</code></td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR.NS is 0.</td>
</tr>
<tr>
<td><code>0b10</code></td>
<td>Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td><code>0b11</code></td>
<td>Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NSID, bits [1:0]**

Non-secure Invasive Debug.

<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>0b00</code></td>
<td>Not implemented. EL3 is not implemented or the Effective value of SCR_EL3.NS is 0.</td>
</tr>
<tr>
<td><code>0b10</code></td>
<td>Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td><code>0b11</code></td>
<td>Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the DBGAUTHSTATUS**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGAUTHSTATUS;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGAUTHSTATUS;
  endif
elsif PSTATE.EL == EL3 then
  return DBGAUTHSTATUS;
endif
Debug Breakpoint Control Registers, n = 0 - 15

The DBGBCR<n> characteristics are:

**Purpose**

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>. If EL2 is implemented and this breakpoint supports Context matching, DBGBVR<n> can be associated with a Breakpoint Extended Value Register DBGBXVR<n> for VMID matching.

**Configuration**

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBCR<n>_EL1[31:0].

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to External register DBGBCR<n>_EL1[31:0].

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGBCR<n> is a 32-bit register.

**Field descriptions**

The DBGBCR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BT | LBN | SSC | HMC | RES0 | BAS | RES0 | PMC | E |

When the E field is zero, all the other fields in the register are ignored.

**Bits [31:24]**

Reserved, RES0.

**BT, bits [23:20]**

Breakpoint Type. Possible values are:
### Table: DBGBCR<\text{n}>, Debug Breakpoint Control Registers, \text{n} = 0 - 15

<table>
<thead>
<tr>
<th>BT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Unlinked instruction address match. (\text{DBGBVR}&lt;\text{n}&gt;) is the address of an instruction.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As (0b0000) with linking enabled.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Unlinked Context ID match. When ARMv8.1-VHE is implemented, EL2 is using AArch64, and the Effective value of (\text{HCR_EL2.E2H}) is 1, if either the PE is executing at EL0 with (\text{HCR_EL2.TGE}) set to 0 or the PE is executing at EL2, then (\text{DBGBVR}&lt;\text{n}&gt;).ContextID must match the (\text{CONTEXTIDR_EL2}) value. Otherwise (\text{DBGBVR}&lt;\text{n}&gt;).ContextID must match the (\text{CONTEXTIDR}) value.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As (0b0010) with linking enabled.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Unlinked instruction address mismatch. (\text{DBGBVR}&lt;\text{n}&gt;) is the address of an instruction to be stepped.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As (0b0100) with linking enabled.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Unlinked (\text{CONTEXTIDR_EL1}) match. (\text{DBGBVR}&lt;\text{n}&gt;) ContextID is a Context ID compared against (\text{CONTEXTIDR}).</td>
</tr>
<tr>
<td>0b0111</td>
<td>As (0b0110) with linking enabled.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Unlinked VMID match. (\text{DBGBXVR}&lt;\text{n}&gt;).VMID is a VMID compared against (\text{VTTBR}.)VMID.</td>
</tr>
<tr>
<td>0b1001</td>
<td>As (0b1000) with linking enabled.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Unlinked VMID and Context ID match. (\text{DBGBXVR}&lt;\text{n}&gt;).ContextID is a Context ID compared against (\text{CONTEXTIDR}), and (\text{DBGBXVR}&lt;\text{n}&gt;).VMID is a VMID compared against (\text{VTTBR}.)VMID.</td>
</tr>
<tr>
<td>0b1011</td>
<td>As (0b1010) with linking enabled.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Unlinked (\text{CONTEXTIDR_EL2}) match. (\text{DBGBXVR}&lt;\text{n}&gt;).ContextID2 is a Context ID compared against (\text{CONTEXTIDR_EL2}).</td>
</tr>
<tr>
<td>0b1101</td>
<td>As (0b1100) with linking enabled.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Unlinked Full Context ID match. (\text{DBGVR}&lt;\text{n}&gt;).ContextID is compared against (\text{CONTEXTIDR}), and (\text{DBGBXVR}&lt;\text{n}&gt;).ContextID2 is compared against (\text{CONTEXTIDR_EL2}).</td>
</tr>
<tr>
<td>0b1111</td>
<td>As (0b1110) with linking enabled.</td>
</tr>
</tbody>
</table>

For more information on Breakpoints and their constraints, see 'Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2.9 and 'Reserved DBGBCR<\text{n}>.BT values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

### LBN, bits [19:16]

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an **UNKNOWN** value.

This field is ignored when the value of DBGBCR<\text{n}>.E is 0.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

### SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint \text{n} is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the \{HMC, SSC, PMC\} fields.
For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, and 'Reserved DBGBCR<n>.{SSC, HMC, PMC} values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**HMC, bit [13]**

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the SSC, bits [15:14] description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**Bits [12:9]**

Reserved, RES0.

**BAS, bits [8:5]**

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Match instruction at</th>
<th>Constraint for debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0011</td>
<td>DBGBVR&lt;n&gt;</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1100</td>
<td>DBGBVR&lt;n&gt;+2</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1111</td>
<td>DBGBVR&lt;n&gt;</td>
<td>Use for A32 instructions</td>
</tr>
</tbody>
</table>

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For more information on using the BAS field in Address Match breakpoints, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Step instruction at</th>
<th>Constraint for debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>-</td>
<td>Use for a match anywhere breakpoint</td>
</tr>
<tr>
<td>0b0011</td>
<td>DBGBVR&lt;n&gt;</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1100</td>
<td>DBGBVR&lt;n&gt;+2</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1111</td>
<td>DBGBVR&lt;n&gt;</td>
<td>Use for A32 instructions</td>
</tr>
</tbody>
</table>

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For more information on using the BAS field in address mismatch breakpoints, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Context matching breakpoints, this field is RES1 and ignored.
The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**Bits [4:3]**

Reserved, RES0.

**PMC, bits [2:1]**

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the DBGBCR<n>.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**E, bit [0]**

Enable breakpoint DBGBVR<n>. Possible values are:

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Breakpoint disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Breakpoint enabled.</td>
</tr>
</tbody>
</table>

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**Accessing the DBGBCR<n>**

Accesses to this register use the following encodings:

$$MRC\langle c\rangle\langle q\rangle \langle coproc\rangle, \langle#\rangle\langle opc1\rangle, \langle Rt\rangle, \langle CRn\rangle, \langle CRm\rangle, \langle#\rangle\langle opc2\rangle$$

<table>
<thead>
<tr>
<th>Source</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>n[3:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>

Page 2261
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
else
  return DBGBCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
else
  return DBGBCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
else
  return DBGBCR[UInt(CRm<3:0>)];
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
else
  DBGBCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
else
  DBGBCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
else
  DBGBCR[UInt(CRm<3:0>)] = R[t];
end if;

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbe63431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The DBGBVR<n> characteristics are:

**Purpose**

Holds a value for use in breakpoint matching, either the virtual address of an instruction or a context ID. Forms breakpoint n together with control register DBGBCR<n>. If EL2 is implemented and this breakpoint supports Context matching, DBGBVR<n> can be associated with a Breakpoint Extended Value Register DBGBXVR<n> for VMID matching.

**Configuration**

AArch32 System register DBGBVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBVR<n> EL1[31:0].

AArch32 System register DBGBVR<n> bits [31:0] are architecturally mapped to External register DBGBVR<n> EL1[31:0].

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

How this register is interpreted depends on the value of DBGBCR<n>.BT.

- When DBGBCR<n>.BT is 0b0x0x, this register holds a virtual address.
- When DBGBCR<n>.BT is 0bxx1x, this register holds a Context ID.

For other values of DBGBCR<n>.BT, this register is RES0.

Some breakpoints might not support Context ID comparison. For more information, see the description of the DBGDIR CTX_CMPs field.

**Field descriptions**

The DBGBVR<n> bit assignments are:

**When DBGBCR<n>.BT == 0b0x0x:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**VA[31:2]**

Bits[31:2] of the address value for comparison.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.
Bits [1:0]

Reserved, RES0.

When DBGBCR<n>.BT == 0b001x:

ContextID

ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against CONTEXTIDR_EL2 when all of the following are true:

- ARMv8.1-VHE is implemented.
- HCR_EL2.{E2H, TGE} is {1,1}.
- The PE is executing at EL0.
- EL2 is enabled in the current Security state, and is using AArch64.

Otherwise, the value is compared against CONTEXTIDR.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

When DBGBCR<n>.BT == 0b101x and HaveEL(EL2):

ContextID

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

When DBGBCR<n>.BT == 0bx11x, HaveEL(EL2) and ARMv8.1-VHE is implemented:

ContextID

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

Accessing the DBGBVR<n>

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCLR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHyptException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCLR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt SoftwareAccess);
    else
        return DBGBVR[UInt(CRm<3:0>)];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCLR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt SoftwareAccess);
    else
        return DBGBVR[UInt(CRm<3:0>)];
    end
elsif PSTATE.EL == EL3 then
    if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt SoftwareAccess);
    else
        return DBGBVR[UInt(CRm<3:0>)];
    end

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCLR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHyptException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCLR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt SoftwareAccess);
    else
        DBGBVR[UInt(CRm<3:0>)] = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCLR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt SoftwareAccess);
    else
        DBGBVR[UInt(CRm<3:0>)] = R[t];
    end
elsif PSTATE.EL == EL3 then
    if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt SoftwareAccess);
    else
        DBGBVR[UInt(CRm<3:0>)] = R[t];
    end
The DBGBXVR<n> characteristics are:

**Purpose**

Holds a value for use in breakpoint matching, to support VMID matching. Used in conjunction with a control register DBGBCR<n> and a value register DBGBVR<n>, where EL2 is implemented and breakpoint n supports Context matching.

**Configuration**

AArch32 System register DBGBXVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBVR<n>_EL1[63:32].

AArch32 System register DBGBXVR<n> bits [31:0] are architecturally mapped to External register DBGBVR<n>_EL1[63:32].

Accesses to this register are UNDEFINED in any of the following cases:

- Breakpoint n is not implemented.
- Breakpoint n does not support Context matching.
- EL2 is not implemented.

For more information, see the description of the DBGDIDR.CTX_CMPs field.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

How this register is interpreted depends on the value of DBGBCR<n>.BT.

- When DBGBCR<n>.BT is 0b10xx, this register holds a VMID.
- When DBGBCR<n>.BT is 0b11xx, this register holds a Context ID.

For other values of DBGBCR<n>.BT, this register is RES0.

**Field descriptions**

The DBGBXVR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bits 31:16</th>
<th>VMID[15:8]</th>
<th>VMID[7:0]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**When DBGBCR<n>.BT == 0b10xx and HaveEL(EL2):**

Reserved, RES0.

**VMID[15:8], bits [15:8]**

When ARMv8.1-VMID16 is implemented:

Extension to VMID[7:0]. See VMID[7:0] for more details.
The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**Otherwise:**

Reserved, RES0.

**VMID[7:0], bits [7:0]**

VMID value for comparison.

The VMID is 8 bits in the following cases.

- EL2 is using AArch32.
- ARMv8.1-VMID16 is not implemented.

When ARMv8.1-VMID16 is implemented and EL2 is using AArch64, it is IMPLEMENTATION DEFINED whether the VMID is 8 bits or 16 bits.

VMID[15:8] is RES0 if any of the following applies:

- The implementation has an 8-bit VMID.
- VTCR_EL2.VS has a value of 0.
- EL2 is using AArch32.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**When DBGBCR<n>.BT == 0b11xx and HaveEL(EL2):**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**ContextID2, bits [31:0]**

When ARMv8.1-VHE is implemented:

Context ID value for comparison against CONTEXTIDR_EL2.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**Otherwise:**

Reserved, RES0.

**Accessing the DBGBXVR<n>**

Accesses to this register use the following encodings:

```
MRC{<c>{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGBXVR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGBXVR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGBXVR[UInt(CRm<3:0>)];

MCR{<c>}{<q>}{<coproc>}, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0001</td>
<td>n[3:0]</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBXVR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBXVR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGBXVR[UInt(CRm<3:0>)] = R[t];
DBGCLAIMCLR, Debug CLAIM Tag Clear register

The DBGCLAIMCLR characteristics are:

**Purpose**

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

**Note**

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMSET register.

**Configuration**

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMCLR_EL1[31:0].

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to External register DBGCLAIMCLR_EL1[31:0].

An implementation must include eight CLAIM tag bits.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally *UNKNOWN* values.

**Attributes**

DBGCLAIMCLR is a 32-bit register.

**Field descriptions**

The DBGCLAIMCLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>RAZ/SBZ</td>
</tr>
<tr>
<td>7-0</td>
<td>CLAIM</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RAZ/SBZ. Software can rely on these bits reading as zero, and must use a should-be-zero policy on writes. Implementations must ignore writes.

**CLAIM, bits [7:0]**

Read or clear CLAIM tag bits. Reading this field returns the current value of the CLAIM tag bits.

Writing a 1 to one of these bits clears the corresponding CLAIM tag bit to 0. This is an indirect write to the CLAIM tag bits. A single write operation can clear multiple CLAIM tag bits to 0.

Writing 0 to one of these bits has no effect.

On a Cold reset, this field resets to 0.
Accessing the DBGCLAIMCLR

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>, \{#\}<opc2>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.<TDA> == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGCLAIMCLR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGCLAIMCLR;
elsif PSTATE.EL == EL3 then
  return DBGCLAIMCLR;

MCR\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>, \{#\}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DebuggerClaimCLR = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DebuggerClaimCLR = R[t];
elsif PSTATE.EL == EL3 then
  DebuggerClaimCLR = R[t];
The DBGCLAIMSET characteristics are:

**Purpose**

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

**Note**

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMCLR register.

**Configuration**

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[31:0].

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to External register DBGCLAIMSET_EL1[31:0].

An implementation must include eight CLAIM tag bits.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGCLAIMSET is a 32-bit register.

**Field descriptions**

The DBGCLAIMSET bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RAZ/SBZ | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| CLAIM    | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Bits [31:8]**

Reserved, RAZ/SBZ. Software can rely on these bits reading as zero, and must use a should-be-zero policy on writes. Implementations must ignore writes.

**CLAIM, bits [7:0]**

Set CLAIM tag bits.

This field is RAO.

Writing a 1 to one of these bits sets the corresponding CLAIM tag bit to 1. This is an indirect write to the CLAIM tag bits. A single write operation can set multiple CLAIM tag bits to 1.

Writing 0 to one of these bits has no effect.

On a Cold reset, this field resets to 0.
## Accessing the DBGCLAIMSET

Accesses to this register use the following encodings:

\[
\text{MRC} \{<c>\}{<q}> \text{coproc}, \{#\}\text{opc1}, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{#\}\text{opc2}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
if EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elsif EL2Enabled() && ELUsingAAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
AArch32.TakeHypTrapException(0x05);
elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
AArch64.AArch32SystemAccessTrap(EL3, 0x05);
else
return DBGCLAIMSET;
elsif PSTATE.EL == EL2 then
if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
AArch64.AArch32SystemAccessTrap(EL3, 0x05);
else
return DBGCLAIMSET;
elsif PSTATE.EL == EL3 then
return DBGCLAIMSET;

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
if EL2Enabled() && !ELUsingAAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elsif EL2Enabled() && ELUsingAAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
AArch32.TakeHypTrapException(0x05);
elsif HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
AArch64.AArch32SystemAccessTrap(EL3, 0x05);
else
DBGCLAIMSET = R[t];
elsif PSTATE.EL == EL2 then
if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TDA == '1' then
AArch64.AArch32SystemAccessTrap(EL3, 0x05);
else
DBGCLAIMSET = R[t];
elsif PSTATE.EL == EL3 then
DBGCLAIMSET = R[t];
DBGDCINT, DCC Interrupt Enable Register

The DBGDCINT characteristics are:

**Purpose**

Enables interrupt requests to be signaled based on the DCC status flags.

**Configuration**

AArch32 System register DBGDCINT bits [31:0] are architecturally mapped to AArch64 System register MDCCINT_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally *UNKNOWN* values.

**Attributes**

DBGDCINT is a 32-bit register.

**Field descriptions**

The DBGDCINT bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>RX, DCC interrupt request enable control for DTRRX. Enables a common COMMIRQ interrupt request to be signaled based on the DCC status flags.</td>
</tr>
<tr>
<td>29</td>
<td>TX, DCC interrupt request enable control for DTRTX. Enables a common COMMIRQ interrupt request to be signaled based on the DCC status flags.</td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bit [31]**

Reserved, RES0.

**RX, bit [30]**

DCC interrupt request enable control for DTRRX. Enables a common COMMIRQ interrupt request to be signaled based on the DCC status flags.

<table>
<thead>
<tr>
<th>RX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt request generated by DTRRX.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt request will be generated on RXfull == 1.</td>
</tr>
</tbody>
</table>

If legacy COMMRX and COMMTX signals are implemented, then these are not affected by the value of this bit.

On a Warm reset, this field resets to 0.

**TX, bit [29]**

DCC interrupt request enable control for DTRTX. Enables a common COMMIRQ interrupt request to be signaled based on the DCC status flags.

<table>
<thead>
<tr>
<th>TX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt request generated by DTRTX.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt request will be generated on TXfull == 0.</td>
</tr>
</tbody>
</table>

If legacy COMMRX and COMMTX signals are implemented, then these are not affected by the value of this bit.

On a Warm reset, this field resets to 0.
Bits [28:0]

Reserved, RES0.

Accessing the DBGDCCINT

Accesses to this register use the following encodings:

\[
\text{MCR\{<c>\}{<q>\} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{{, \{#\}<opc2>}}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDCCINT;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDCCINT;
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return DBGDCCINT;
    endif
else
    return DBGDCCINT;
endif

MCR\{<c>\}{<q>\} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{{, \{#\}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
    AArch32.TakeHypTrapException(0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGDCCINT = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGDCCINT = R[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
    AArch32.TakeMonitorTrapException();
  else
    DBGDCCINT = R[t];
The DBGDEVID characteristics are:

**Purpose**

Adds to the information given by the `DBGDIDR` by describing other features of the debug implementation.

**Configuration**

This register is required in all implementations.

**Attributes**

DBGDEVID is a 32-bit register.

**Field descriptions**

The DBGDEVID bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>CIDMask</td>
</tr>
<tr>
<td>28</td>
<td>AuxRegs</td>
</tr>
<tr>
<td>25</td>
<td>DoubleLock</td>
</tr>
<tr>
<td>22</td>
<td>VirtExtns</td>
</tr>
<tr>
<td>19</td>
<td>VectorCatch</td>
</tr>
<tr>
<td>16</td>
<td>BPAddrMask</td>
</tr>
<tr>
<td>13</td>
<td>WPAddrMask</td>
</tr>
<tr>
<td>10</td>
<td>PCSample</td>
</tr>
</tbody>
</table>

**CIDMask, bits [31:28]**

Indicates the level of support for the Context ID matching breakpoint masking capability. Permitted values of this field are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Context ID masking is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Context ID masking is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved. The value of this for Armv8 is 0b0000.

**AuxRegs, bits [27:24]**

Indicates support for Auxiliary registers. Permitted values for this field are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for External Debug Auxiliary Control Register, <code>EDACR</code>.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**DoubleLock, bits [23:20]**

OS Double Lock implemented. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>OS Double Lock is not implemented. <code>DBGOSDLR</code> is RAZ/ WI.</td>
</tr>
<tr>
<td>0b0001</td>
<td>OS Double Lock is implemented. <code>DBGOSDLR</code> is RW.</td>
</tr>
</tbody>
</table>

ARMv8.0-DoubleLock implements the functionality identified by the value 0b0001.

All other values are reserved.
VirtExtns, bits [19:16]

Indicates whether EL2 is implemented. Permitted values of this field are:

<table>
<thead>
<tr>
<th>VirtExtns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL2 is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL2 is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

VectorCatch, bits [15:12]

Defines the form of Vector Catch exception implemented. Permitted values of this field are:

<table>
<thead>
<tr>
<th>VectorCatch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Address matching Vector Catch exception implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Exception matching Vector Catch exception implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

BPAddrMask, bits [11:8]

Indicates the level of support for the instruction address matching breakpoint masking capability. Permitted values of this field are:

<table>
<thead>
<tr>
<th>BPAddrMask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Breakpoint address masking might be implemented. If not implemented, DBGBCR&lt;n&gt;[28:24] is RAZ/WI.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Breakpoint address masking is implemented.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Breakpoint address masking is not implemented. DBGBCR&lt;n&gt;[28:24] is RES0.</td>
</tr>
</tbody>
</table>

All other values are reserved. The value of this for Armv8 is 0b1111.

WPAddrMask, bits [7:4]

Indicates the level of support for the data address matching watchpoint masking capability. Permitted values of this field are:

<table>
<thead>
<tr>
<th>WPAddrMask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Watchpoint address masking might be implemented. If not implemented, DBGWCR&lt;n&gt;.MASK (Address mask) is RAZ/WI.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Watchpoint address masking is implemented.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Watchpoint address masking is not implemented. DBGWCR&lt;n&gt;.MASK (Address mask) is RES0.</td>
</tr>
</tbody>
</table>

All other values are reserved. The value of this for Armv8 is 0b0001.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers. Permitted values of this field are:

<table>
<thead>
<tr>
<th>PCSample</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PC Sample-based Profiling Extension is not implemented in the external debug registers space.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Only EDPCSR and EDCIDSR are implemented. This option is only permitted if EL3 and EL2 are not implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>EDPCSR, EDCIDSR, and EDVIDSR are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When ARMv8.2-PCSample is implemented, the only permitted value is 0b0000.
Note

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

Accessing the DBGDEVID

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & !ELUsingAArch32(EL2) & MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() & ELUsingAArch32(EL2) & HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) & !ELUsingAArch32(EL3) & MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGDEVID;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) & !ELUsingAArch32(EL3) & MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGDEVID;
elsif PSTATE.EL == EL3 then
  return DBGDEVID;
DBGDEVID1, Debug Device ID register 1

The DBGDEVID1 characteristics are:

**Purpose**

Adds to the information given by the DBGDIDR by describing other features of the debug implementation.

**Configuration**

This register is required in all implementations.

**Attributes**

DBGDEVID1 is a 32-bit register.

**Field descriptions**

The DBGDEVID1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
<th>Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>RES0</td>
<td>PCSROffset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**PCSROffset, bits [3:0]**

This field indicates the offset applied to PC samples returned by reads of EDPCSR. Permitted values of this field in Armv8 are:

<table>
<thead>
<tr>
<th>PCSROffset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EDPCSR is not implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EDPCSR implemented. Samples have no offset applied and do not sample the instruction set state in AArch32 state.</td>
</tr>
</tbody>
</table>

When Armv8.2-PCSample is implemented, the only permitted value is 0b0000.

**Note**

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

**Accessing the DBGDEVID1**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c}\}{<q}> \text{ <coproc}, \{#<opc1>, <Rt>, <CRn>, <CRm>,\{#<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b0001</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDEVID1;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDEVID1;
    end
elsif PSTATE.EL == EL3 then
    return DBGDEVID1;
The DBGDEVID2 characteristics are:

**Purpose**

Reserved for future descriptions of features of the debug implementation.

**Configuration**

**Attributes**

DBGDEVID2 is a 32-bit register.

**Field descriptions**

The DBGDEVID2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RES0.

**Accessing the DBGDEVID2**

Accesses to this register use the following encodings:

\[
\text{MRC}\{\langle c\rangle}\{\langle q\rangle\} \langle\text{coproc}\rangle, \{\#\}\langle\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle\{, \{\#\}\langle\text{opc2}\rangle\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0000</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then  
  UNDEFINED;  
elsif PSTATE.EL == EL1 then  
  if EL2Enabled() &amp; !ELUsingAArch32(EL2) &amp; MDCR_EL2.&lt;TDE,TDA&gt; != '00' then  
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);  
  elseif EL2Enabled() &amp; ELUsingAArch32(EL2) &amp; HDCR.&lt;TDE,TDA&gt; != '00' then  
    AArch32.TakeHypTrapException(0x05);  
  elseif HaveEL(EL3) &amp; !ELUsingAArch32(EL3) &amp; MDCR_EL3.TDA == '1' then  
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);  
  else  
    return DBGDEVID2;  
elsif PSTATE.EL == EL2 then  
  if HaveEL(EL3) &amp; !ELUsingAArch32(EL3) &amp; MDCR_EL3.TDA == '1' then  
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);  
  else  
    return DBGDEVID2;  
elsif PSTATE.EL == EL3 then  
  return DBGDEVID2;
DBGIDR, Debug ID Register

The DBGIDR characteristics are:

**Purpose**

Specifies which version of the Debug architecture is implemented, and some features of the debug implementation.

**Configuration**

If EL1 cannot use AArch32 then the implementation of this register is *optional* and deprecated.

**Attributes**

DBGIDR is a 32-bit register.

**Field descriptions**

The DBGIDR bit assignments are:

```
   31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
| WRPs | BRPs | CTX_CMPs | Version | RES1 | nSUHD_imp | RES0 | SE_imp | RES0 |
```

**WRPs, bits [31:28]**

The number of watchpoints implemented, minus 1.

Permitted values of this field are from \texttt{0b0001} for 2 implemented watchpoints, to \texttt{0b1111} for 16 implemented watchpoints.

The value of \texttt{0b0000} is reserved.

If AArch64 is implemented, this field has the same value as \texttt{ID_AA64DFR0_EL1.WRPs}.

**BRPs, bits [27:24]**

The number of breakpoints implemented, minus 1.

Permitted values of this field are from \texttt{0b0001} for 2 implemented breakpoint, to \texttt{0b1111} for 16 implemented breakpoints.

The value of \texttt{0b0000} is reserved.

If AArch64 is implemented, this field has the same value as \texttt{ID_AA64DFR0_EL1.BRPs}.

**CTX_CMPs, bits [23:20]**

The number of breakpoints that can be used for Context matching, minus 1.

Permitted values of this field are from \texttt{0b0000} for 1 Context matching breakpoint, to \texttt{0b1111} for 16 Context matching breakpoints.

The Context matching breakpoints must be the highest addressed breakpoints. For example, if six breakpoints are implemented and two are Context matching breakpoints, they must be breakpoints 4 and 5.

If AArch64 is implemented, this field has the same value as \texttt{ID_AA64DFR0_EL1.CTX_CMPs}.
**Version, bits [19:16]**

The Debug architecture version. Defined values are:

<table>
<thead>
<tr>
<th>Version</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Armv6, v6 Debug architecture.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Armv6, v6.1 Debug architecture.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Armv7, v7 Debug architecture, with baseline CP14 registers implemented.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Armv7, v7 Debug architecture, with all CP14 registers implemented.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Armv7, v7.1 Debug architecture.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Armv8, v8 Debug architecture.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv8.1, v8 Debug architecture, with Virtualization Host Extensions.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Armv8.2, v8.2 Debug architecture.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Armv8.4, v8.4 Debug architecture.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In any Armv8 implementation, the values 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted.

- If ARMv8.1-VHE is not implemented, the only permitted value is 0b0110.
- In an Armv8.0 implementation, the value 0b1000 or higher is not permitted.

**Bit [15]**

Reserved, RES1.

**nSUHD_imp, bit [14]**

In Armv7-A, was Secure User Halting Debug not implemented.

The value of this bit must match the value of the SE_imp bit.

**Bit [13]**

Reserved, RES0.

**SE_imp, bit [12]**

EL3 implemented. The meanings of the values of this bit are:

<table>
<thead>
<tr>
<th>SE_imp</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL3 not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL3 implemented.</td>
</tr>
</tbody>
</table>

The value of this bit must match the value of the nSUHD_imp bit.

**Bits [11:0]**

Reserved, RES0.

**Accessing the DBGDIDR**

Arm deprecates any access to this register from EL0.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x05);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x05);
  end if
elsif ELUsingAArch32(EL1) && DBGDSCRExt.UDCCdis == '1' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
    AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  end if
elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
  AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
  AArch32.TakeHypTrapException(0x05);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x05);
else
  return DBGDIDR;
else if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGDIDR;
else if PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGDIDR;
else if PSTATE.EL == EL3 then
  return DBGDIDR;
end if

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGDRAR, Debug ROM Address Register

The DBGDRAR characteristics are:

**Purpose**

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Armv8 deprecates any use of this register.

**Configuration**

AArch32 System register DBGDRAR bits [63:0] are architecturally mapped to AArch64 System register MDRAR_EL1[63:0].

If EL1 cannot use AArch32 then the implementation of this register is **OPTIONAL** and deprecated.

**Attributes**

DBGDRAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, bits [31:0] are read.

**Field descriptions**

The DBGDRAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>57</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>46</td>
<td>ROMADDR[47:12]</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Valid</td>
</tr>
</tbody>
</table>

**Bits [63:48]**

Reserved, RES0.

**ROMADDR[47:12], bits [47:12]**

Bits[47:12] of the ROM table physical address.

If the physical address size in bits (PAsize) is less than 48 then the register bits corresponding to ROMADDR [47:PAsize] are RES0.

Bits [11:0] of the ROM table physical address are zero.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system that supports AArch32 at the highest implemented Exception level.

In an implementation that includes EL3, ROMADDR is an address in Non-secure memory. It is **IMPLEMENTATION DEFINED** whether the ROM table is also accessible in Secure memory.

**Bits [11:2]**

Reserved, RES0.

**Valid, bits [1:0]**

This field indicates whether the ROM Table address is valid. The permitted values of this field are:
Valid | Meaning
--- | ---
0b00 | ROM Table address is not valid. Software must ignore ROMADDR.
0b11 | ROM Table address is valid.

Other values are reserved.

**Accessing the DBGDRAR**

Accesses to this register use the following encodings:

**MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}**

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
AArch64.AArch32SystemAccessTrap(EL2, 0x05);
else
AArch64.AArch32SystemAccessTrap(EL1, 0x05);
elif ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1' then
if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
AArch32.TakeHypTrapException(0x00);
else
UNDEFINED;
elif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
AArch32.TakeHypTrapException(0x05);
else
return DBGDRAR<31:0>;
eelse PSTATE.EL == EL1 then
if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
AArch32.TakeHypTrapException(0x05);
else
return DBGDRAR<31:0>;
elif PSTATE.EL == EL2 then
return DBGDRAR<31:0>;
elif PSTATE.EL == EL3 then
return DBGDRAR<31:0>;
e

**MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>**

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b0001</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x0C);
    elsif ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
        AArch32.TakeHypTrapException(0x0C);
    else
        return DBGDRAR;
    else
        PSTATE.EL == EL1 then
            if EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL2.<TDE,TDRA> != '00' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
                AArch32.TakeHypTrapException(0x0C);
            else
                return DBGDRAR;
            end
        elsif PSTATE.EL == EL2 then
            return DBGDRAR;
        elsif PSTATE.EL == EL3 then
            return DBGDRAR;
        end
    end
end
DBGDSAR, Debug Self Address Register

The DBGDSAR characteristics are:

**Purpose**

In earlier versions of the Arm Architecture, this register defines the offset from the base address defined in DBGDRAR of the physical base address of the debug registers for the PE. Armv8 deprecates any use of this register.

**Configuration**

If EL1 cannot use AArch32 then the implementation of this register is **OPTIONAL** and deprecated.

**Attributes**

DBGDSAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, bits [31:0] are read.

**Field descriptions**

The DBGDSAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RAZ.</td>
</tr>
<tr>
<td>60</td>
<td>Reserved, RAZ.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Bits [63:2]**

Reserved, RES0.

**Bits [1:0]**

Reserved, RAZ.

This field indicates whether the debug self address offset is valid. For ARMv8, this field is always 0b00, the offset is not valid.

**Accessing the DBGDSAR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#<opc1>, <Rt>, <CRn>, <CRm>{, {#<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        end if
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    else
        return DBGDSAR<31:0>;
    end if
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
            AArch32.TakeHypTrapException(0x05);
        else
            return DBGDSAR<31:0>;
        end if
    elsif PSTATE.EL == EL2 then
        return DBGDSAR<31:0>;
    elsif PSTATE.EL == EL3 then
        return DBGDSAR<31:0>;
    end if
end if

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b0010</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
    AArch64.AArch32SystemAccessTrap(EL1, 0x0C);
  else if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
  else if ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else UNDEFINED;
  else if EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> != '00') then
    AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
    AArch32.TakeHypTrapException(0x0C);
  else return DBGDSAR;
else if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL1.<TDE,TDRA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
    AArch32.TakeHypTrapException(0x0C);
  else return DBGDSAR;
elsif PSTATE.EL == EL2 then
  return DBGDSAR;
elsif PSTATE.EL == EL3 then
  return DBGDSAR;
**DBGDCRext, Debug Status and Control Register, External View**

The DBGDCRext characteristics are:

**Purpose**

Main control register for the debug implementation.

**Configuration**

AArch32 System register DBGDCRext bits [31:0] are architecturally mapped to AArch64 System register MDSCR_EL1[31:0].

AArch32 System register DBGDCRext bits [15:2] are architecturally mapped to AArch32 System register DBGDCRint[15:2].

This register is required in all implementations.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGDCRext is a 32-bit register.

**Field descriptions**

The DBGDCRext bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| TFO | RXfull | TXfull | RES0 | RXOTXU | RES0 | INTdis | TDA | RES0 | SC2 | NS | SPNIDdis | SPIDdis | MDBGen | HDRES0 | UDCCdis | RES0 | ERROR |

**TFO, bit [31]**

*When ARMv8.4-Trace is implemented:*

Trace Filter override. Used for save/restore of EDSCR.TFO.

When the OS Lock is unlocked, DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When the OS Lock is locked, DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.TFO. Reads and writes of this bit are indirect accesses to EDSCR.TFO.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

**Otherwise:**

Reserved, RES0.

**RXfull, bit [30]**

DTRRX full. Used for save/restore of EDSCR.RXfull.
When \texttt{DBGOSLSR.OSLK} == 0, software must treat this bit as UNK/SBZP.

When \texttt{DBGOSLSR.OSLK} == 1, this bit holds the value of \texttt{EDSCR.RXfull}. Reads and writes of this bit are indirect accesses to \texttt{EDSCR.RXfull}.

Arm deprecates use of this bit other than for save/restore. Use \texttt{DBGDSCRint} to access the DTRRX full status.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When \texttt{DBGOSLSR.OSLK} == 1, access to this field is \texttt{RW}.
- When \texttt{DBGOSLSR.OSLK} == 0, access to this field is \texttt{RO}.

**TXfull, bit [29]**

DTRTX full. Used for save/restore of \texttt{EDSCR.TXfull}.

When \texttt{DBGOSLSR.OSLK} == 0, software must treat this bit as UNK/SBZP.

When \texttt{DBGOSLSR.OSLK} == 1, this bit holds the value of \texttt{EDSCR.TXfull}. Reads and writes of this bit are indirect accesses to \texttt{EDSCR.TXfull}.

Arm deprecates use of this bit other than for save/restore. Use \texttt{DBGDSCRint} to access the DTRTX full status.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When \texttt{DBGOSLSR.OSLK} == 1, access to this field is \texttt{RW}.
- When \texttt{DBGOSLSR.OSLK} == 0, access to this field is \texttt{RO}.

**Bit [28]**

Reserved, RES0.

**RXO, bit [27]**

Used for save/restore of \texttt{EDSCR.RXO}.

When \texttt{DBGOSLSR.OSLK} == 0, software must treat this bit as UNK/SBZP.

When \texttt{DBGOSLSR.OSLK} == 1, this bit holds the value of \texttt{EDSCR.RXO}. Reads and writes of this bit are indirect accesses to \texttt{EDSCR.RXO}.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When \texttt{DBGOSLSR.OSLK} == 1, access to this field is \texttt{RW}.
- When \texttt{DBGOSLSR.OSLK} == 0, access to this field is \texttt{RO}.

**TXU, bit [26]**

Used for save/restore of \texttt{EDSCR.TXU}.

When \texttt{DBGOSLSR.OSLK} == 0, software must treat this bit as UNK/SBZP.

When \texttt{DBGOSLSR.OSLK} == 1, this bit holds the value of \texttt{EDSCR.TXU}. Reads and writes of this bit are indirect accesses to \texttt{EDSCR.TXU}.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When \texttt{DBGOSLSR.OSLK} == 1, access to this field is \texttt{RW}.
- When \texttt{DBGOSLSR.OSLK} == 0, access to this field is \texttt{RO}. 
Bits [25:24]
Reserved, RES0.

INTdis, bits [23:22]
Used for save/restore of EDSCR.INTdis.
When DBGOSLRSR.OSLK == 0, this field is RO, and software must treat it as UNK/SBZP.
When DBGOSLRSR.OSLK == 1, this field is RW and holds the value of EDSCR.INTdis. Reads and writes of this field are indirect accesses to EDSCR.INTdis.
The architected behavior of this field determines the value it returns after a reset.
Accessing this field has the following behavior:
- When DBGOSLRSR.OSLK == 1, access to this field is RW.
- When DBGOSLRSR.OSLK == 0, access to this field is RO.

TDA, bit [21]
Used for save/restore of EDSCR.TDA.
When DBGOSLRSR.OSLK == 0, software must treat this bit as UNK/SBZP.
When DBGOSLRSR.OSLK == 1, this bit holds the value of EDSCR.TDA. Reads and writes of this bit are indirect accesses to EDSCR.TDA.
The architected behavior of this field determines the value it returns after a reset.
Accessing this field has the following behavior:
- When DBGOSLRSR.OSLK == 1, access to this field is RW.
- When DBGOSLRSR.OSLK == 0, access to this field is RO.

Bit [20]
Reserved, RES0.

SC2, bit [19]
When ARMv8.0-PCSample is implemented, ARMv8.1-VHE is implemented and ARMv8.2-PCSample is not implemented:
Used for save/restore of EDSCR.SC2.
When DBGOSLRSR.OSLK == 0, software must treat this bit as UNK/SBZP.
When DBGOSLRSR.OSLK == 1, this bit holds the value of EDSCR.SC2. Reads and writes of this bit are indirect accesses to EDSCR.SC2.
Accessing this field has the following behavior:
- When DBGOSLRSR.OSLK == 1, access to this field is RW.
- When DBGOSLRSR.OSLK == 0, access to this field is RO.

Otherwise:
Reserved, RES0.

NS, bit [18]
Non-secure status. Returns the inverse of IsSecure().
Arm deprecates use of this field.

Access to this field is **RO**.

**SPNIDdis, bit [17]**

*When HaveEL(EL3):*

Secure privileged profiling disabled status bit.

<table>
<thead>
<tr>
<th>SPNIDdis</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Profiling allowed in Secure privileged modes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Profiling prohibited in Secure privileged modes.</td>
</tr>
</tbody>
</table>

This field reads as 0 if any of the following applies, and reads as 1 otherwise:

- ARMv8.2-Debug is not implemented and ExternalSecureNoninvasiveDebugEnabled() returns TRUE.
- EL3 is using AArch32 and the value of SDCR.SPME is 1.
- EL3 is using AArch64 and the value of MDCR_EL3.SPME is 1.

Arm deprecates use of this field.

Access to this field is **RO**.

**Otherwise:**

Reserved, RES0.

**SPIDdis, bit [16]**

*When HaveEL(EL3):*

Secure privileged AArch32 invasive self-hosted debug disabled status bit. The value of this bit depends on the value of SDCR.SPD and the pseudocode function AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled().

<table>
<thead>
<tr>
<th>SPIDdis</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Self-hosted debug enabled in Secure privileged AArch32 modes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Self-hosted debug disabled in Secure privileged AArch32 modes.</td>
</tr>
</tbody>
</table>

This bit reads as 1 if any of the following is true and reads as 0 otherwise:

- EL3 is using AArch32 and SDCR.SPD has the value 0b10.
- EL3 is using AArch64 and MDCR_EL3.SPD32 has the value 0b10.
- EL3 is using AArch32, SDCR.SPD has the value 0b00, and AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled() returns FALSE.
- EL3 is using AArch64, MDCR_EL3.SPD32 has the value 0b00, and AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled() returns FALSE.

Arm deprecates use of this field.

Access to this field is **RO**.

**Otherwise:**

Reserved, RES0.

**MDBGen, bit [15]**

Monitor debug events enable. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.
### MDBGen

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

### HDE, bit [14]

Used for save/restore of EDSCR.HDE.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.HDE. Reads and writes of this bit are indirect accesses to EDSCR.HDE.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When DBGOSLSR.OSLK == 1, access to this field is **RW**.
- When DBGOSLSR.OSLK == 0, access to this field is **RO**.

### Bit [13]

Reserved, RES0.

### UDCCdis, bit [12]

Traps EL0 accesses to the DCC registers to Undefined mode.

<table>
<thead>
<tr>
<th>UDCCdis</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped. EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIIDR, DBGDSAR, and DBGDRAR are trapped to Undefined mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped. EL0 accesses to the DBGDSCRint, DBGDTRRXint, DBGDTRTXint, DBGDIIDR, DBGDSAR, and DBGDRAR are trapped to Undefined mode.</td>
</tr>
</tbody>
</table>

**Note**

All accesses to these registers are trapped, including LDC and STC accesses to DBGDTRTXint and DBGDTRRXint, and MRRC accesses to DBGDSAR and DBGDRAR.

Traps of EL0 accesses to the DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

On a Warm reset, this field resets to 0.

### Bits [11:7]

Reserved, RES0.

### ERR, bit [6]

Used for save/restore of EDSCR.ERR.

When DBGOSLSR.OSLK == 0, software must treat this bit as UNK/SBZP.

When DBGOSLSR.OSLK == 1, this bit holds the value of EDSCR.ERR. Reads and writes of this bit are indirect accesses to EDSCR.ERR.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:
• When DBGOSLSR.OSLK == 1, access to this field is **RW**.
• When DBGOSLSR.OSLK == 0, access to this field is **RO**.

### MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

<table>
<thead>
<tr>
<th>MOE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Breakpoint.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Software breakpoint (BKPT) instruction.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Vector catch.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Watchpoint.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

### Bits [1:0]

Reserved, **RES0**.

### Accessing the DBGDSCRext

Individual fields within this register might have restricted accessibility when the OS lock is unlocked, **DBGOSLSR.OSLK** == 0. See the field descriptions for more detail.

Accesses to this register use the following encodings:

```plaintext
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGDSCRext;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
      return DBGDSCRext;
  elsif PSTATE.EL == EL3 then
    return DBGDSCRext;
```

```plaintext
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
    endif
elsif PSTATE.EL == EL3 then
    DBGDSCRext = R[t];
The DBGDSCRint characteristics are:

**Purpose**

Main control register for the debug implementation. This is an internal, read-only view.

**Configuration**

AArch32 System register DBGDSCRint bits [30:29] are architecturally mapped to AArch64 System register MDCCSR_EL0[30:29].

AArch32 System register DBGDSCRint bits [15:2] are architecturally mapped to AArch32 System register DBGDSCRext[15:2].

This register is required in all implementations.

DBGDSCRint.{NS, SPNIDdis, SPIDis, MDBGen, UDCCdis, MOE} are **UNKNOWN** when the register is accessed at EL0. However, although these values are not accessible at EL0 by instructions that are neither **UNPREDICTABLE** nor return **UNKNOWN** values, it is permissible for an implementation to return the values of DBGDSCRext.{NS, SPNIDdis, SPIDis, MDBGen, UDCCdis, MOE} for these fields at EL0.

It is also permissible for an implementation to return the same values as defined for a read of DBGDSCRint at EL1 or above. (This is the case even if the implementation does not support AArch32 at EL1 or above.)

**Attributes**

DBGDSCRint is a 32-bit register.

**Field descriptions**

The DBGDSCRint bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RXfull | TXfull | RES0 | NS | SPNIDdis | SPIDis | MDBGen | RES0 | UDCCdis | RES0 | MOE | RES0 |

**Bit [31]**

Reserved, RES0.

**RXfull, bit [30]**

DTRRX full. Read-only view of the equivalent bit in the EDSRC.

**TXfull, bit [29]**

DTRTX full. Read-only view of the equivalent bit in the EDSRC.

**Bits [28:19]**

Reserved, RES0.
NS, bit [18]

Non-secure status.
Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

SPNIDdis, bit [17]

Secure privileged non-invasive debug disable.
Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

SPIDdis, bit [16]

Secure privileged invasive debug disable.
Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

MDBGen, bit [15]

Monitor debug events enable.
Read-only view of the equivalent bit in the DBGDSCRext.

Bits [14:13]

Reserved, RES0.

UDCCdis, bit [12]

User mode access to Debug Communications Channel disable.
Read-only view of the equivalent bit in the DBGDSCRext. Arm deprecates use of this field.

Bits [11:6]

Reserved, RES0.

MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

<table>
<thead>
<tr>
<th>MOE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0b0011</td>
<td>Software breakpoint (BKPT) instruction</td>
</tr>
<tr>
<td>0b0101</td>
<td>Vector catch</td>
</tr>
<tr>
<td>0b1010</td>
<td>Watchpoint</td>
</tr>
</tbody>
</table>

Read-only view of the equivalent bit in the DBGDSCRext.

Bits [1:0]

Reserved, RES0.

Accessing the DBGDSCRint

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    else
        UNDEFINED;
    elseif EL2Enabled() && ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRint;
else if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRint;
else if PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRint;
else if PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return DBGDSCRint;
DBGDTRRXext, Debug OS Lock Data Transfer Register, Receive, External View

The DBGDTRRXext characteristics are:

**Purpose**

Used for save/restore of DBGDTRRXint. It is a component of the Debug Communications Channel.

**Configuration**

AArch32 System register DBGDTRRXext bits [31:0] are architecturally mapped to AArch64 System register OSDTRRX_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGDTRRXext is a 32-bit register.

**Field descriptions**

The DBGDTRRXext bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Update DTRRX without side-effect</td>
</tr>
</tbody>
</table>

*Update DTRRX without side-effect.*

Writes to this register update the value in DTRRX and do not change RXfull.

Reads of this register return the last value written to DTRRX and do not change RXfull.

For the full behavior of the Debug Communications Channel, see ‘The Debug Communication Channel and Instruction Transfer Register’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter H4.

This field resets to an architecturally UNKNOWN value.

**Accessing the DBGDTRRXext**

Arm deprecates reads and writes of DBGDTRRXext through the System register interface when the OS Lock is unlocked, DBGOSLSR.OSLK == 0.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
    AArch32.TakeHypTrapException(0x05);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
    AArch32.TakeMonitorTrapException();
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  else
    return DBGDTRRXext;
else
  elsif PSTATE.EL == EL2 then
    if EL3Enabled() && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif EL3Enabled() && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
      AArch32.TakeMonitorTrapException();
    elsif EL3Enabled() && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
      return DBGDTRRXext;
else
  elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
      AArch32.TakeMonitorTrapException();
    else
      return DBGDTRRXext;
else
  return DBGDTRRXext;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRRXext = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
            AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGDTRRXext = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
            AArch32.TakeMonitorTrapException();
        else
            DBGDTRRXext = R[t];
    else
        DBGDTRRXext = R[t];
DBGDTRRXint, Debug Data Transfer Register, Receive

The DBGDTRRXint characteristics are:

**Purpose**

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See Arch64-DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

**Configuration**

AArch32 System register DBGDTRRXint bits [31:0] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0].

AArch32 System register DBGDTRRXint bits [31:0] are architecturally mapped to External register DBGDTRRX_EL0[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGDTRRXint is a 32-bit register.

**Field descriptions**

The DBGDTRRXint bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Update DTRRX |

**Bits [31:0]**

Update DTRRX.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an **UNKNOWN** value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see The Debug Communication Channel and Instruction Transfer Register.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the DBGDTRRXint**

Data can be stored to memory from this register using STC.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

| coproc | opc1 | CRn | CRm | opc2 |
if PSTATE.EL == EL0 then
    if !Halted() && !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elsif !Halted() && !ELUsingAArch32(EL1) && DBGDSCRext.UCDCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch32.TakeHypTrapException(0x00);
        endif
    elsif !Halted() && !ELUsingAArch32(EL1) && DBGDSCRext.UCDCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch32.TakeHypTrapException(0x00);
        endif
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x05);
        else
            UNDEFINED;
        endelsif
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRRXint;
    endelsif
else
    if PSTATE.EL == EL1 then
        if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDSCR_EL2.TDCC == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        endif
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRRXint;
    endelsif
else
    if PSTATE.EL == EL2 then
        if !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDSCR_EL3.TDCC == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        endif
    elsif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRRXint;
    endelsif
else
    if PSTATE.EL == EL3 then
        if !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return DBGDTRRXint;
        endelsif
    else
        return DBGDTRRXint;
    endelsif
    return DBGDTRRXint;
endelif
DBGDTRTXext, Debug OS Lock Data Transfer Register, Transmit

The DBGDTRTXext characteristics are:

**Purpose**

Used for save/restore of DBGDTRTXint. It is a component of the Debug Communication Channel.

**Configuration**

AArch32 System register DBGDTRTXext bits [31:0] are architecturally mapped to AArch64 System register OSDTRTX_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGDTRTXext is a 32-bit register.

**Field descriptions**

The DBGDTRTXext bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Return DTRTX without side-effect.</td>
</tr>
<tr>
<td>30</td>
<td>Return DTRTX without side-effect.</td>
</tr>
<tr>
<td>29</td>
<td>Returns of this register return the value in DTRTX and do not change TXfull.</td>
</tr>
<tr>
<td>28</td>
<td>Writes of this register update the value in DTRTX and do not change TXfull.</td>
</tr>
<tr>
<td>27</td>
<td>For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter H4.</td>
</tr>
<tr>
<td>26</td>
<td>On a Cold reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
</tbody>
</table>

**Accessing the DBGDTRTXext**

Arm deprecates reads and writes of DBGDTRTXext through the System register interface when the OS Lock is unlocked. DBGOSLSR.OSLK == 0.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TD> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TD> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRTXext;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRTXext;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return DBGDTRTXext;

MCR{<c>}{<q>}{<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    else
        DBGDTRTXext = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTXext = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRTXext = R[t];
The DBGDTRTXint characteristics are:

**Purpose**

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See DBGDTRX_EL0 for additional architectural mappings. It is a component of the Debug Communication Channel.

**Configuration**

AArch32 System register DBGDTRTXint bits [31:0] are architecturally mapped to AArch64 System register DBGDTRX_EL0[31:0].

AArch32 System register DBGDTRTXint bits [31:0] are architecturally mapped to External register DBGDTRX_EL0[31:0].

RW fields in this register reset to architecturally \textit{UNKNOWN} values.

**Attributes**

DBGDTRTXint is a 32-bit register.

**Field descriptions**

The DBGDTRTXint bit assignments are:

\[
\begin{array}{cccccccccccccccccccccc}
\hline
\end{array}
\]

Return DTRTX

**Bits [31:0]**

Return DTRTX.

Writes to this register:

- If TXfull is set to 1, set DTRTX to \textit{UNKNOWN}.
- If TXfull is set to 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see The Debug Communication Channel and Instruction Transfer Register.

On a Cold reset, this field resets to an architecturally \textit{UNKNOWN} value.

**Accessing the DBGDTRTXint**

Data can be loaded from memory into this register using LDC (immediate) and LDC (literal).

Accesses to this register use the following encodings:

\[
\text{MCR}\{<c>\}{<q>}\ <\text{coproc}, \ {#}\{<opc1>, \ <Rt>, \ <CRn>, \ <CRm>{, \ {#}\{<opc2>}}
\]

\[
\begin{array}{cccccc}
\text{coproc} & \text{opc1} & \text{CRn} & \text{CRm} & \text{opc2} \\
\end{array}
\]
if PSTATE.EL == EL0 then
    if !Halted() && !ELUsingAArch32(EL1) && MDCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        endif
    else
        UNDEFINED;
    endif
    elsif !Halted() && ELUsingAArch32(EL1) && DBGDSCRext.UCCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        endif
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRTXint = R[t];
    endif
elsif PSTATE.EL == EL1 then
    if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && HCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTXint = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif !Halted() && EL2Enabled() && MDCR_EL2.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRTXint = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    else
        DBGDTRTXint = R[t];
    endif
else
    DBGDTRTXint = R[t];
endelse
if PSTATE.EL == EL0 then
    if !Halted() && !ELUsingAArch32(EL1) && MDCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        endif
    else
        UNDEFINED;
    endif
    elsif !Halted() && ELUsingAArch32(EL1) && DBGDSCRext.UCCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        endif
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elseif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTXint = R[t];
    endif
elsif PSTATE.EL == EL1 then
    if !Halted() && EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && MDCR_EL2.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif !Halted() && EL2Enabled() && ELUsingAArch32(EL2) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRTXint = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif !Halted() && HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    elsif !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTXint = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if !Halted() && HaveEL(EL3) && !ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else
        DBGDTRTXint = R[t];
    endif
else
    DBGDTRTXint = R[t];
endelse
DBGOSDLR, Debug OS Double Lock Register

The DBGOSDLR characteristics are:

**Purpose**

Locks out the external debug interface.

**Configuration**

AArch32 System register DBGOSDLR bits [31:0] are architecturally mapped to AArch64 System register OSDLR_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGOSDLR is a 32-bit register.

**Field descriptions**

The DBGOSDLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

**Bits [31:1]**

Reserved, RES0.

**DLK, bit [0]**

When ARMv8.0-DoubleLock is implemented:

OS Double Lock control bit.

<table>
<thead>
<tr>
<th>DLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>OS Double Lock unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>OS Double Lock locked, if DBGPRCR, CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RAZ/WI.

**Accessing the DBGOSDLR**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
    (IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' &&
    (IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by HDCR.TDOSA") then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
    (IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGOSDLR;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
    (IsFeatureImplemented("ARMv8.0-DoubleLock") || boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGOSDLR = R[t];
  end
elsif PSTATE.EL == EL3 then
  DBGOSDLR = R[t];
else
  DBGOSDLR = R[t];
end

The DBGOSECCR characteristics are:

**Purpose**

Provides a mechanism for an operating system to access the contents of EDECCR that are otherwise invisible to software, so it can save/restore the contents of EDECCR over powerdown on behalf of the external debugger.

**Configuration**

AArch32 System register DBGOSECCR bits [31:0] are architecturally mapped to AArch64 System register OSECCR_EL1[31:0].

AArch32 System register DBGOSECCR bits [31:0] are architecturally mapped to External register EDECCR[31:0].

If DBGOSLSR.OSLK == 0 then DBGOSECCR returns an UNKNOWN value on reads and ignores writes.

**Attributes**

DBGOSECCR is a 32-bit register.

**Field descriptions**

The DBGOSECCR bit assignments are:

**When DBGOSLSR.OSLK == 1:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>EDECCR</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**EDECCR, bits [31:0]**

Used for save/restore to EDECCR over powerdown.

Reads or writes to this field are indirect accesses to EDECCR.

**Accessing the DBGOSECCR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGSOECCR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGSOECCR;
        elsif PSTATE.EL == EL3 then
            return DBGSOECCR;
        end
    end
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSECCR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGOSECCR = R[t];
        elsif PSTATE.EL == EL3 then
            DBGOSECCR = R[t];
        end
    end
end
DBGOSLAR, Debug OS Lock Access Register

The DBGOSLAR characteristics are:

**Purpose**

Provides a lock for the debug registers. The OS Lock also disables some debug exceptions and debug events.

**Configuration**

AArch32 System register DBGOSLAR bits [31:0] are architecturally mapped to AArch64 System register OSLAR_EL1[31:0].

AArch32 System register DBGOSLAR bits [31:0] are architecturally mapped to External register OSLAR_EL1[31:0].

**Attributes**

DBGOSLAR is a 32-bit register.

**Field descriptions**

The DBGOSLAR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

OS Lock Access, bits [31:0]

OS Lock Access. Writing the value 0xC5ACCE55 to the DBGOSLAR sets the OS lock to 1. Writing any other value sets the OS lock to 0.

Use DBGOSLRSR.OSLK to check the current status of the lock.

**Accessing the DBGOSLAR**

Accesses to this register use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSLAR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGOSLAR = R[t];
    end
elsif PSTATE.EL == EL3 then
    DBGOSLAR = R[t];

DBGOSLAR, Debug OS Lock Access Register

27/09/2019 18:22; 6134483bd14dc8c12a99c984c9be3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGOSLRS, Debug OS Lock Status Register

The DBGOSLRS characteristics are:

**Purpose**

Provides status information for the OS Lock.

**Configuration**

AArch32 System register DBGOSLRS bits [31:0] are architecturally mapped to AArch64 System register OSLSR_EL1[31:0].

The OS Lock status is also visible in the external debug interface through EDPRSR.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGOSLRS is a 32-bit register.

**Field descriptions**

The DBGOSLRS bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>OSLM[1]</td>
<td>nTT</td>
<td>OSLK</td>
<td>OSLM[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**OSLM[1], bit [3]**

This field is bit[1] of OSLM[1:0].

OS lock model implemented. Identifies the form of OS save and restore mechanism implemented.

<table>
<thead>
<tr>
<th>OSLM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>OS Lock not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>OS Lock implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

The OSLM field is split as follows:

- OSLM[1] is DBGOSLRS[3].
- OSLM[0] is DBGOSLRS[0].

**nTT, bit [2]**

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.

**OSLK, bit [1]**

OS Lock Status. The possible values are:
### OSLK

<table>
<thead>
<tr>
<th>OSLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>OS Lock unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>OS Lock locked.</td>
</tr>
</tbody>
</table>

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The following resets apply:

- On a Cold reset, this field resets to 1.
- On a Warm reset, the value of this field is unchanged.

#### OSLM[0], bit [0]

This field is bit[0] of OSLM[1:0].

See OSLM[1] for the field description.

### Accessing the DBGOSLSR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \text{ <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2>}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```{}
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGOSLSR;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGOSLSR;
  end
elsif PSTATE.EL == EL3 then
  return DBGOSLSR;
```

---

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGPRCR, Debug Power Control Register

The DBGPRCR characteristics are:

**Purpose**

Controls behavior of the PE on powerdown request.

**Configuration**

AArch32 System register DBGPRCR bits [31:0] are architecturally mapped to AArch64 System register DBGPRCR_EL1[31:0].

Bit [0] of this register is mapped to EDPRCR.CORENPDRQ, bit [0] of the external view of this register.

The other bits in these registers are not mapped to each other.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DBGPRCR is a 32-bit register.

**Field descriptions**

The DBGPRCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>CORENPDRQ</td>
</tr>
</tbody>
</table>

**Bits [31:1]**

Reserved, RES0.

**CORENPDRQ, bit [0]**

*When ARMv8.3-DoPD is implemented:*

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the **IMPLEMENTATION DEFINED** nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>CORENPDRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the system responds to a powerdown request, it powers down Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.</td>
</tr>
</tbody>
</table>

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is **IMPLEMENTATION DEFINED** whether this bit is reset to the Cold reset value on exit from an **IMPLEMENTATION DEFINED** software-visible retention state. For more information about retention states see Core power domain power states.

**Note**
Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The following resets apply:

- On a Cold reset, this field is set to 1 if the powerup request is implemented and the powerup request has been asserted, and is set to 0 otherwise.
- On a Warm reset, the value of this field is unchanged.

**Otherwise:**

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>CORENPDRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the system responds to a powerdown request, it powers down Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.</td>
</tr>
</tbody>
</table>

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of EDPRCR COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see Core power domain power states.

**Note**

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The following resets apply:

- On a Cold reset, this field resets to the value in EDPRCR COREPURQ.
- On a Warm reset, the value of this field is unchanged.

**Accessing the DBGPRCR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \ <\text{coproc}>, \ {#}<\text{opc1}>, \ <\text{Rt}>, \ <\text{CRn}>, \ <\text{CRm}>\{, \ {#}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGPRCR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGPRCR;
elsif PSTATE.EL == EL3 then
  return DBGPRCR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGPRCR = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGPRCR = R[t];
elsif PSTATE.EL == EL3 then
  DBGPRCR = R[t];
**DBGVCR, Debug Vector Catch Register**

The DBGVCR characteristics are:

**Purpose**

Controls Vector Catch debug events.

**Configuration**

AArch32 System register DBGVCR bits [31:0] are architecturally mapped to AArch64 System register DBGVCR32_EL2[31:0].

This register is required in all implementations.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGVCR is a 32-bit register.

**Field descriptions**

The DBGVCR bit assignments are:

**When HaveEL(EL3) and ELUsingAArch32(EL3):**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NSF | NSI | RES0 | NSD | NSP | NSS | NSU | RES0 | MF | MI | RES0 | MD | MP | MS | RES0 | SF | SI | RES0 | SD | SP | SS | SU | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 |

**NSF, bit [31]**

FIQ vector catch enable in Non-secure state.

The exception vector offset is 0x1C.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSI, bit [30]**

IRQ vector catch enable in Non-secure state.

The exception vector offset is 0x18.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bit [29]**

Reserved, RES0.

**NSD, bit [28]**

Data Abort vector catch enable in Non-secure state.

The exception vector offset is 0x10.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is 0x0C.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is 0x08.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is 0x04.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:16]

Reserved, RES0.

MF, bit [15]

FIQ vector catch enable in Monitor mode.

The exception vector offset is 0x1C.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

MI, bit [14]

IRQ vector catch enable in Monitor mode.

The exception vector offset is 0x18.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

MD, bit [12]

Data Abort vector catch enable in Monitor mode.

The exception vector offset is 0x10.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

MP, bit [11]

Prefetch Abort vector catch enable in Monitor mode.

The exception vector offset is 0x0C.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
**MS, bit [10]**

Secure Monitor Call (SMC) vector catch enable in Monitor mode.

The exception vector offset is 0x08.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [9:8]**

Reserved, **RES0**.

**SF, bit [7]**

FIQ vector catch enable in Secure state.

The exception vector offset is 0x1C.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**SI, bit [6]**

IRQ vector catch enable in Secure state.

The exception vector offset is 0x18.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bit [5]**

Reserved, **RES0**.

**SD, bit [4]**

Data Abort vector catch enable in Secure state.

The exception vector offset is 0x10.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**SP, bit [3]**

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is 0x0C.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**SS, bit [2]**

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is 0x08.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**SU, bit [1]**

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is 0x04.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
Bit [0]

Reserved, RES0.

**When HaveEL(EL3) and !ELUsingAArch32(EL3):**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NSF | NSI | RES | NSD | NSP | NSS | NSU | RES | SF | SI | RES | SD | SP | SS | SU | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES | RES |

**NSF, bit [31]**

FIQ vector catch enable in Non-secure state.
The exception vector offset is 0x1C.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSI, bit [30]**

IRQ vector catch enable in Non-secure state.
The exception vector offset is 0x18.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Bit [29]**

Reserved, RES0.

**NSD, bit [28]**

Data Abort vector catch enable in Non-secure state.
The exception vector offset is 0x10.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSP, bit [27]**

Prefetch Abort vector catch enable in Non-secure state.
The exception vector offset is 0x0C.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSS, bit [26]**

Supervisor Call (SVC) vector catch enable in Non-secure state.
The exception vector offset is 0x08.
On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSU, bit [25]**

Undefined Instruction vector catch enable in Non-secure state.
The exception vector offset is 0x04.
On a Warm reset, this field resets to an architecturally UNKNOWN value.
Bits [24:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.
The exception vector offset is 0x1C.
On a Warm reset, this field resets to an architecturally unknown value.

SI, bit [6]

IRQ vector catch enable in Secure state.
The exception vector offset is 0x18.
On a Warm reset, this field resets to an architecturally unknown value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort vector catch enable in Secure state.
The exception vector offset is 0x10.
On a Warm reset, this field resets to an architecturally unknown value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.
The exception vector offset is 0x0C.
On a Warm reset, this field resets to an architecturally unknown value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.
The exception vector offset is 0x08.
On a Warm reset, this field resets to an architecturally unknown value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.
The exception vector offset is 0x04.
On a Warm reset, this field resets to an architecturally unknown value.

Bit [0]

Reserved, RES0.
When !HaveEL(EL3):

<table>
<thead>
<tr>
<th>Bits [31:8]</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F, bit [7]</td>
<td>FIQ vector catch enable. The exception vector offset is 0x1C. On a Warm reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>I, bit [6]</td>
<td>IRQ vector catch enable. The exception vector offset is 0x18. On a Warm reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>Bit [5]</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>D, bit [4]</td>
<td>Data Abort vector catch enable. The exception vector offset is 0x10. On a Warm reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>P, bit [3]</td>
<td>Prefetch Abort vector catch enable. The exception vector offset 0x0C. On a Warm reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>S, bit [2]</td>
<td>Supervisor Call (SVC) vector catch enable. The exception vector offset is 0x08. On a Warm reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>U, bit [1]</td>
<td>Undefined Instruction vector catch enable. The exception vector offset is 0x04. On a Warm reset, this field resets to an architecturally UNKNOWN value.</td>
</tr>
</tbody>
</table>

DBGVCR, Debug Vector Catch Register
Bit [0]

Reserved, RES0.

**Accessing the DBGVCR**

Accesses to this register use the following encodings:

\[ \text{MRC}\{<c>\}{<q>} \text{ <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2>} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGVCR;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGVCR = R[t];
    end
elsif PSTATE.EL == EL3 then
    DBGVCR = R[t];

MCR{<c>{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>{, \{#\}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGVCR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGVCR = R[t];
    end
elsif PSTATE.EL == EL3 then
    DBGVCR = R[t];
DBGWCR<n>, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR<n> characteristics are:

**Purpose**

Holds control information for a watchpoint. Forms watchpoint n together with value register DBGWVR<n>.

**Configuration**

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGWCR<n>_EL1[31:0].

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to External register DBGWCR<n>_EL1[31:0].

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DBGWCR<n> is a 32-bit register.

**Field descriptions**

The DBGWCR<n> bit assignments are:

```
+---------------+---------------+---------------+---------------+---------------+---------------+---------------+---------------+---------------+
<p>| 31            | 30            | 29            | 28            | 27            | 26            | 25            | 24            | 23            |</p>
<table>
<thead>
<tr>
<th>RES0</th>
<th>MASK</th>
<th>RES0</th>
<th>WT</th>
<th>LBN</th>
<th>SSC</th>
<th>HMC</th>
<th>BAS</th>
<th>LSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>ME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
<td>ME</td>
</tr>
</tbody>
</table>
```

When the E field is zero, all the other fields in the register are ignored.

**Bits [31:29]**

Reserved, RES0.

**MASK, bits [28:24]**

Address mask. Only objects up to 2GB can be watched using a single mask.

<table>
<thead>
<tr>
<th>MASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>No mask.</td>
</tr>
<tr>
<td>0b00001</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

If programmed with a reserved value, a watchpoint must behave as if either:

- MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCRn_EL1.
- The watchpoint is disabled.

Software must not rely on this property because the behavior of reserved values might change in a future revision of the architecture.

Other values mask the corresponding number of address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b111111 masking 31 address bits (0x7FFFFFFF mask for address).
The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**Bits [23:21]**

Reserved, RES0.

**WT, bit [20]**

Watchpoint type. Possible values are:

<table>
<thead>
<tr>
<th>WT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unlinked data address match.</td>
</tr>
<tr>
<td>0b1</td>
<td>Linked data address match.</td>
</tr>
</tbody>
</table>

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**LBN, bits [19:16]**

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**SSC, bits [15:14]**

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, and 'Reserved DBGCR<n>.{SSC, HMC, PMC} values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.

**HMC, bit [13]**

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, the value of this field is unchanged.
**BAS, bits [12:5]**

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by DBGWVR<\text{n}>, is being watched.

<table>
<thead>
<tr>
<th>BAS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0bxxxxxxx1</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;</td>
</tr>
<tr>
<td>0bxxxxxx1x</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+1</td>
</tr>
<tr>
<td>0bxxxxxx1xx</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+2</td>
</tr>
<tr>
<td>0bxxxxxxx1xx</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+3</td>
</tr>
</tbody>
</table>

In cases where DBGWVR<\text{n}> addresses a double-word:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Description, if DBGWVR&lt;\text{n}&gt;[2] == 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0bxxx1xxxx</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+4</td>
</tr>
<tr>
<td>0bx1xxxxxx</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+5</td>
</tr>
<tr>
<td>0bx1xxxxxx</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+6</td>
</tr>
<tr>
<td>0b1xxxxxxx</td>
<td>Match byte at DBGWVR&lt;\text{n}&gt;+7</td>
</tr>
</tbody>
</table>


The valid values for BAS are non-zero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<\text{n}>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**LSC, bits [4:3]**

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

<table>
<thead>
<tr>
<th>LSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>Match instructions that load from a watchpointed address.</td>
</tr>
<tr>
<td>0b10</td>
<td>Match instructions that store to a watchpointed address.</td>
</tr>
<tr>
<td>0b11</td>
<td>Match instructions that load from or store to a watchpointed address.</td>
</tr>
</tbody>
</table>

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**PAC, bits [2:1]**

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint \text{n} is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.
E, bit [0]

Enable watchpoint n. Possible values are:

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint enabled.</td>
</tr>
</tbody>
</table>

The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**Accessing the DBGWCR<n>**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>}{<q}\}} \text{ <coproc>, \#<opc1>, <Rt>, <CRn>, <CRm>{, 
\#<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWCR[UInt(CRm<3:0>)];
  elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWCR[UInt(CRm<3:0>)];
  elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    return DBGWCR[UInt(CRm<3:0>)];

MCR\{<c>}{<q}\} <coproc>, \#<opc1>, <Rt>, <CRn>, <CRm>{, \#<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elseif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWCR[UInt(CRm<3:0>)] = R[t];
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elseif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWCR[UInt(CRm<3:0>)] = R[t];
  end
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWCR[UInt(CRm<3:0>)] = R[t];
DBGWFAR, Debug Watchpoint Fault Address Register

The DBGWFAR characteristics are:

**Purpose**

Previously returned information about the address of the instruction that accessed a watchpointed address. Is now deprecated and RES0.

**Configuration**

**Attributes**

DBGWFAR is a 32-bit register.

**Field descriptions**

The DBGWFAR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RES0.

**Accessing the DBGWFAR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
elif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGWFAR;
elif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGWFAR;
elif PSTATE.EL == EL3 then
  return DBGWFAR;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGWFAR = R[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    DBGWFAR = R[t];
  endif
elsif PSTATE.EL == EL3 then
  DBGWFAR = R[t];

27/09/2019 18:22; 6134483bd14dc8c12a99c984c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The DBGWVR<n> characteristics are:

### Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register DBGWCR<n>.

### Configuration

AArch32 System register DBGWVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGWVR<n>_EL1[31:0].

AArch32 System register DBGWVR<n> bits [31:0] are architecturally mapped to External register DBGWVR<n>_EL1[31:0].

If watchpoint n is not implemented then accesses to this register are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

### Attributes

DBGWVR<n> is a 32-bit register.

### Field descriptions

The DBGWVR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | **VA** |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | **RES0** |

**VA, bits [31:2]**

Bits[31:2] of the address value for comparison.


The following resets apply:

- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
- On a Warm reset, the value of this field is unchanged.

**Bits [1:0]**

Reserved, **RES0**.

### Accessing the DBGWVR<n>

Accesses to this register use the following encodings:
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGWVR[UInt(CRm<3:0>)];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWVR[UInt(CRm<3:0>)] = R[t];
  endif
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWVR[UInt(CRm<3:0>)] = R[t];
  endif
else
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
    AArch32.TakeHypTrapException(0x05);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  else
    return DBGWVR[UInt(CRm<3:0>)];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
  elsif ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWVR[UInt(CRm<3:0>)] = R[t];
  endif
elsif PSTATE.EL == EL3 then
  if ELUsingAArch32(EL1) && DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
    Halt(DebugHalt_SoftwareAccess);
  else
    DBGWVR[UInt(CRm<3:0>)] = R[t];
  endif
else
end
DCCIMVAC, Data Cache line Clean and Invalidate by VA to PoC

The DCCIMVAC characteristics are:

**Purpose**

Clean and Invalidate data or unified cache line by virtual address to PoC.

**Configuration**

AArch32 System instruction DCCIMVAC performs the same function as AArch64 System instruction DCIVAC.

**Attributes**

DCCIMVAC is a 32-bit System instruction.

**Field descriptions**

The DCCIMVAC input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Virtual address to use

**Bits [31:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DCCIMVAC instruction**

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instruction (DC*)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TPC == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DCCIMVAC(R[t]);
    elsif PSTATE.EL == EL2 then
        DCCIMVAC(R[t]);
    elsif PSTATE.EL == EL3 then
        DCCIMVAC(R[t]);

The DCCISW characteristics are:

**Purpose**

Clean and Invalidate data or unified cache line by set/way.

**Configuration**

AArch32 System instruction DCCISW performs the same function as AArch64 System instruction DC_CISW.

**Attributes**

DCCISW is a 32-bit System instruction.

**Field descriptions**

The DCCISW input value bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = Log2(ASSOCIATIVITY), L = Log2(LINELEN), B = (L + S), S = Log2(NSETS).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.

**Executing the DCCISW instruction**

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c}\}\{<q}\} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TSW == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DCCISW(R[t]);
    endif
else
    DCCISW(R[t]);
endif
```

27/09/2019 18:22; 6134483bd14dc8c12a99e984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DCCMVAC, Data Cache line Clean by VA to PoC

The DCCMVAC characteristics are:

**Purpose**

Clean data or unified cache line by virtual address to PoC.

**Configuration**

AArch32 System instruction DCCMVAC performs the same function as AArch64 System instruction DC CVAC.

**Attributes**

DCCMVAC is a 32-bit System instruction.

**Field descriptions**

The DCCMVAC input value bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>Virtual address to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>30</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>29</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>28</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>27</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>26</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>25</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>24</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>23</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>22</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>21</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>20</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>19</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>18</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>17</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>16</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>15</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>14</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>13</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>12</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>11</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>10</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>9</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>8</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>7</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>6</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>5</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>4</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>3</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>2</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>1</td>
<td>Virtual address to use</td>
</tr>
<tr>
<td>0</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DCCMVAC instruction**

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instruction (DC*)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
      UNDEFINED;
elsif PSTATE.EL == EL1 then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
         AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
         AArch32.TakeHypTrapException(0x03);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
         AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TPC == '1' then
         AArch32.TakeHypTrapException(0x03);
      else
         DCCMVAC(R[t]);
      endif
elsif PSTATE.EL == EL2 then
      DCCMVAC(R[t]);
elsif PSTATE.EL == EL3 then
      DCCMVAC(R[t]);
**DCCMVAU, Data Cache line Clean by VA to PoU**

The DCCMVAU characteristics are:

**Purpose**

Clean data or unified cache line by virtual address to PoU.

**Configuration**

AArch32 System instruction DCCMVAU performs the same function as AArch64 System instruction **DC CVAU**.

**Attributes**

DCCMVAU is a 32-bit System instruction.

**Field descriptions**

The DCCMVAU input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Virtual address to use</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DCCMVAU instruction**

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instruction (DC*)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    DCCMVAU(R[t]);
  endif
elsif PSTATE.EL == EL2 then
  DCCMVAU(R[t]);
elsif PSTATE.EL == EL3 then
  DCCMVAU(R[t]);
else
  DCCMVAU(R[t]);
endif
DCCSW, Data Cache line Clean by Set/Way

The DCCSW characteristics are:

**Purpose**

Clean data or unified cache line by set/way.

**Configuration**

AArch32 System instruction DCCSW performs the same function as AArch64 System instruction DC CSW.

**Attributes**

DCCSW is a 32-bit System instruction.

**Field descriptions**

The DCCSW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>SetWay</td>
<td>Contains two fields:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Way, bits[31:32-A], the number of the way to operate on.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Set, bits[B-1:L], the number of the set to operate on.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Bits[L-1:4] are RES0.</td>
</tr>
<tr>
<td></td>
<td>Level</td>
<td>Cache level to operate on, minus 1.</td>
</tr>
<tr>
<td></td>
<td>Bit</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**SetWay, bits [31:4]**

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = \(\log_2(\text{ASSOCIATIVITY})\), \(L = \log_2(\text{LINELEN})\), \(B = (L + S)\), \(S = \log_2(\text{NSETS})\).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

**Level, bits [3:1]**

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

**Bit [0]**

Reserved, RES0.

**Executing the DCCSW instruction**

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.
Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>}\ <\text{coproc}, \{\#}\text{opc1}, <\text{Rt}, <\text{CRn}, <\text{CRm}\{, \{\#}\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() \&\& ELUsingAArch32(EL2) \&\& HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HCR_EL2.TSW == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HCR.TSW == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    DCCISW(R[t]);
  endif
elsif PSTATE.EL == EL2 then
  DCCISW(R[t]);
elsif PSTATE.EL == EL3 then
  DCCISW(R[t]);
**DCIMVAC, Data Cache line Invalidate by VA to PoC**

The DCIMVAC characteristics are:

**Purpose**

Invalidate data or unified cache line by virtual address to PoC.

**Configuration**

AArch32 System instruction DCIMVAC performs the same function as AArch64 System instruction DC IVAC.

**Attributes**

DCIMVAC is a 32-bit System instruction.

**Field descriptions**

The DCIMVAC input value bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Virtual address to use*

**Bits [31:0]**

Virtual address to use. No alignment restrictions apply to this VA.

**Executing the DCIMVAC instruction**

It is **IMPLEMENTATION DEFINED** whether, when this instruction is executed, it can generate a watchpoint. If this instruction can generate a watchpoint this is prioritized in the same way as other watchpoints.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instruction (DC*)' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

\[ \text{MCR}\{<c}\}{<q}\} <\text{coproc}, \{#<\text{opc1}\}, <\text{Rt}, <\text{CRn}, <\text{CRm}\{, \{#<\text{opc2}\} \}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TPC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DCCIMVAC(R[t]);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.<DC,VM> != '00' then
    DCCIMVAC(R[t]);
  else
    DCIMVAC(R[t]);
elsif PSTATE.EL == EL2 then
  DCIMVAC(R[t]);
elsif PSTATE.EL == EL3 then
  DCIMVAC(R[t]);
DCISW, Data Cache line Invalidate by Set/Way

The DCISW characteristics are:

**Purpose**

Invalidate data or unified cache line by set/way.

**Configuration**

AArch32 System instruction DCISW performs the same function as AArch64 System instruction DCISW.

**Attributes**

DCISW is a 32-bit System instruction.

**Field descriptions**

The DCISW input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:3-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

A = \( \log_2(\text{ASSOCIATIVITY}) \), L = \( \log_2(\text{LINELEN}) \), B = (L + S), S = \( \log_2(\text{NSETS}) \).

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

**Executing the DCISW instruction**

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is **UNDEFINED**
- The instruction performs cache maintenance on one of:
  - No cache lines.
  - A single arbitrary cache line.
  - Multiple arbitrary cache lines.
Accesses to this instruction use the following encodings:

\[
\text{MCR}\{c\}\{q\} <\text{coproc},\{#<\text{opc}>,\ <\text{Rt},\ <\text{CRn},\ <\text{CRm}{,\ {#<\text{opc}2}\}}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TSW == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.SWIO == '1' then
    DCISW(R[t]);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<DC,VM> != '00' then
    DCISW(R[t]);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.SWIO == '1' then
    DCISW(R[t]);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.<DC,VM> != '00' then
    DCISW(R[t]);
  else
    DCISW(R[t]);
elsif PSTATE.EL == EL2 then
  DCISW(R[t]);
elsif PSTATE.EL == EL3 then
  DCISW(R[t]);
DFAR, Data Fault Address Register

The DFAR characteristics are:

**Purpose**

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception.

**Configuration**

AArch32 System register DFAR bits [31:0] are architecturally mapped to AArch64 System register FAR_EL1[31:0].

AArch32 System register DFAR bits [31:0] (S) are architecturally mapped to AArch32 System register HDFAR[31:0] when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DFAR is a 32-bit register.

**Field descriptions**

The DFAR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| VA of faulting address of synchronous Data Abort exception |

**Bits [31:0]**

VA of faulting address of synchronous Data Abort exception.

This field resets to an architecturally UNKNOWN value.

**Accessing the DFAR**

Accesses to this register use the following encodings:

\[MRC{\langle c\rangle}{\langle q\rangle} \langle coproc\rangle, \{#\}\langle opc1\rangle, \langle Rt\rangle, \langle CRn\rangle, \langle CRm\rangle, \{#\}\langle opc2\rangle\]\n
<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVRM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVRM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return DFAR_S;
        else
            return DFAR_NS;
        end if;
    else
        return DFAR;
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFAR_NS;
    else
        return DFAR;
    end if;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DFAR_S;
    else
        return DFAR_NS;
    end if;
else
    return DFAR_NS;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVRM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVRM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            DFAR_S = R[t];
        else
            DFAR_NS = R[t];
        end if;
    else
        DFAR = R[t];
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFAR_NS = R[t];
    else
        DFAR = R[t];
    end if;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFAR_S = R[t];
    else
        DFAR_NS = R[t];
    end if;
DFSR, Data Fault Status Register

The DFSR characteristics are:

**Purpose**

Holds status information about the last data fault.

**Configuration**

AArch32 System register DFSR bits [31:0] are architecturally mapped to AArch64 System register ESR_EL1[31:0].

The current translation table format determines which format of the register is used.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

DFSR is a 32-bit register.

**Field descriptions**

The DFSR bit assignments are:

When TTBCR.EAE == 0:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    | RES0 | FnV | AET | CM | Ext | WnR | FS[4] | LPAE | RES0 | Domain | FS[3:0] |

**Bits [31:17]**

Reserved, RES0.

**FnV, bit [16]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>DFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>DFAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Data Abort exceptions.

This field resets to an architecturally UNKNOWN value.

**AET, bits [15:14]**

Asynchronous Error Type. When the RAS Extension is implemented, this field describes the state of the PE after taking an asynchronous Data Abort exception. Possible values are:

<table>
<thead>
<tr>
<th>AET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Uncontainable error (UC) or uncategorized.</td>
</tr>
<tr>
<td>0b01</td>
<td>Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b10</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
<tr>
<td>0b11</td>
<td>Recoverable error (UER).</td>
</tr>
</tbody>
</table>
When the RAS Extension is not implemented, or on a synchronous Data Abort, this field is RES0.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

In the event of multiple errors taken as a single SError interrupt exception, the overall state of the PE is reported.

**Note**

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field resets to an architecturally UNKNOWN value.

### CM, bit [13]

Cache maintenance fault. For synchronous faults, this bit indicates whether a cache maintenance instruction generated the fault. The possible values of this bit are:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort not caused by execution of a cache maintenance instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by execution of a cache maintenance instruction, or on an address translation.</td>
</tr>
</tbody>
</table>

On a synchronous Data Abort on a translation table walk, this bit is UNKNOWN.

On an asynchronous fault, this bit is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

### ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This field resets to an architecturally UNKNOWN value.

### WnR, bit [11]

Write not Read bit. Indicates whether the abort was caused by a write or a read instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort caused by a read instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by a write instruction.</td>
</tr>
</tbody>
</table>

For faults on the cache maintenance and address translation System instructions in the (coproc==0b1111) encoding space this bit always returns a value of 1.

This field resets to an architecturally UNKNOWN value.

### FS[4], bit [10]

This field is bit[4] of FS[4:0].

Fault status bits. Possible values of FS[4:0] are:
<table>
<thead>
<tr>
<th>FS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00001</td>
<td>Alignment fault</td>
</tr>
<tr>
<td>0b00010</td>
<td>Debug exception</td>
</tr>
<tr>
<td>0b00011</td>
<td>Access flag fault, level 1</td>
</tr>
<tr>
<td>0b00100</td>
<td>Fault on instruction cache maintenance</td>
</tr>
<tr>
<td>0b00101</td>
<td>Translation fault, level 1</td>
</tr>
<tr>
<td>0b00110</td>
<td>Access flag fault, level 2</td>
</tr>
<tr>
<td>0b00111</td>
<td>Translation fault, level 2</td>
</tr>
<tr>
<td>0b01000</td>
<td>Synchronous External abort, not on translation table walk</td>
</tr>
<tr>
<td>0b01001</td>
<td>Domain fault, level 1</td>
</tr>
<tr>
<td>0b01011</td>
<td>Domain fault, level 2</td>
</tr>
<tr>
<td>0b01100</td>
<td>Synchronous External abort, on translation table walk, level 1</td>
</tr>
<tr>
<td>0b01101</td>
<td>Permission fault, level 1</td>
</tr>
<tr>
<td>0b01110</td>
<td>Synchronous External abort, on translation table walk, level 2</td>
</tr>
<tr>
<td>0b01111</td>
<td>Permission fault, level 2</td>
</tr>
<tr>
<td>0b10000</td>
<td>TLB conflict abort</td>
</tr>
<tr>
<td>0b10100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown fault)</td>
</tr>
<tr>
<td>0b10101</td>
<td>IMPLEMENTATION DEFINED fault (Unsupported Exclusive access fault)</td>
</tr>
<tr>
<td>0b10110</td>
<td>SError interrupt</td>
</tr>
<tr>
<td>0b11000</td>
<td>SError interrupt, from a parity or ECC error on memory access</td>
</tr>
<tr>
<td>0b11001</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk</td>
</tr>
<tr>
<td>0b11100</td>
<td>Synchronous parity or ECC error on translation table walk, level 1</td>
</tr>
<tr>
<td>0b11110</td>
<td>Synchronous parity or ECC error on translation table walk, level 2</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b11000, 0b11001, 0b11100, and 0b11110, are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The FS field is split as follows:

- FS[4] is DFSR[10].
- FS[3:0] is DFSR[3:0].

This field resets to an architecturally UNKNOWN value.

**LPAE, bit [9]**

On taking a Data Abort exception, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table formats.</td>
</tr>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table formats.</td>
</tr>
</tbody>
</table>

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

This field resets to an architecturally UNKNOWN value.

**Bit [8]**

Reserved, RES0.

**Domain, bits [7:4]**

The domain of the fault address.

Arm deprecates any use of this field, see ‘The Domain field in the DFSR’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
This field is **UNKNOWN** for certain faults where the DFSR is updated and reported using the Short-descriptor FSR encodings, see 'Validity of Domain field on faults that update the DFSR when using the Short-descriptor encodings' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**FS[3:0], bits [3:0]**

This field is bits[3:0] of FS[4:0].


**When TTBCR.EAE == 1:**

|   | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|   | RES0 | FnV | AET | CM | Ext | WnR | RES0 | PAE | Res0 | STATUS |

**Bits [31:17]**

Reserved, **RES0**.

**FnV, bit [16]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>DFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>DFAR is not valid, and holds an <strong>UNKNOWN</strong> value.</td>
</tr>
</tbody>
</table>

This field is only valid for a synchronous External abort other than a synchronous External abort on a translation table walk. It is **RES0** for all other Data Abort exceptions.

This field resets to an architecturally **UNKNOWN** value.

**AET, bits [15:14]**

Asynchronous Error Type. When the RAS Extension is implemented, this field describes the state of the PE after taking an asynchronous Data Abort exception. Possible values are:

<table>
<thead>
<tr>
<th>AET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Uncontainable error (UC) or un categorized.</td>
</tr>
<tr>
<td>0b01</td>
<td>Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b10</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
<tr>
<td>0b11</td>
<td>Recoverable error (UER).</td>
</tr>
</tbody>
</table>

When the RAS Extension is not implemented, or on a synchronous Data Abort, this field is **RES0**.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

In the event of multiple errors taken as a single SError interrupt exception, the overall state of the PE is reported.

**Note**

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field resets to an architecturally **UNKNOWN** value.
CM, bit [13]

Cache maintenance fault. For synchronous faults, this bit indicates whether a cache maintenance instruction generated the fault. The possible values of this bit are:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort not caused by execution of a cache maintenance instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by execution of a cache maintenance instruction.</td>
</tr>
</tbody>
</table>

On a synchronous Data Abort on a translation table walk, this bit is UNKNOWN.

On an asynchronous fault, this bit is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This field resets to an architecturally UNKNOWN value.


Write not Read bit. Indicates whether the abort was caused by a write or a read instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort caused by a read instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by a write instruction.</td>
</tr>
</tbody>
</table>

For faults on the cache maintenance and address translation System instructions in the (coproc==0b1111) encoding space this bit always returns a value of 1.

This field resets to an architecturally UNKNOWN value.

Bit [10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table formats.</td>
</tr>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table formats.</td>
</tr>
</tbody>
</table>

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

This field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:
### Status

<table>
<thead>
<tr>
<th>Status</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault in <a href="#">TTBR0</a> or <a href="#">TTBR1</a>.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b001100</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b010001</td>
<td>SError interrupt.</td>
</tr>
<tr>
<td>0b010100</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011001</td>
<td>SError interrupt, from a parity or ECC error on memory access.</td>
</tr>
<tr>
<td>0b011100</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100000</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b100010</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
<tr>
<td>0b110100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown fault).</td>
</tr>
<tr>
<td>0b110101</td>
<td>IMPLEMENTATION DEFINED fault (Unsupported Exclusive access fault).</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011001, 0b011100, 0b011101, 0b011110, and 0b011111, are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

### Accessing the DFSR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#\}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return DFSR.S;
        else
            return DFSR_NS;
        end
    else
        return DFSR;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFSR_NS;
    else
        return DFSR;
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DFSR.S;
    else
        return DFSR_NS;
    end
else
    if PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TV == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TV == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                DFSR.S = R[t];
            else
                DFSR_NS = R[t];
            end
        end
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            DFSR_NS = R[t];
        else
            DFSR = R[t];
        end
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            DFSR.S = R[t];
        else
            DFSR_NS = R[t];
        end
    end
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TV == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TV == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            DFSR.S = R[t];
        else
            DFSR_NS = R[t];
        end
    else
        DFSR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFSR_NS = R[t];
    else
        DFSR = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFSR.S = R[t];
    else
        DFSR_NS = R[t];
    end
else
    if PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TV == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TV == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                DFSR.S = R[t];
            else
                DFSR_NS = R[t];
            end
        end
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            DFSR_NS = R[t];
        else
            DFSR = R[t];
        end
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            DFSR.S = R[t];
        else
            DFSR_NS = R[t];
        end
    end
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
**DISR, Deferred Interrupt Status Register**

The DISR characteristics are:

**Purpose**
Records that an SError interrupt has been consumed by an ESB instruction.

**Configuration**
AArch32 System register DISR bits [31:0] are architecturally mapped to AArch64 System register DISR_EL1[31:0] when the highest implemented Exception level is using AArch64.

This register is present only when RAS is implemented. Otherwise, direct accesses to DISR are UNDEFINED. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**
DISR is a 32-bit register.

**Field descriptions**
The DISR bit assignments are:

**When the ESB instruction is executed at EL2:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| A  | RES0 | AET | EA | RES0 | DFSC |

**A, bit [31]**
Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

This field resets to an architecturally UNKNOWN value.

**Bits [30:12]**
Reserved, RES0.

**AET, bits [11:10]**
Asynchronous Error Type. See the description of HSR.AET for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

**EA, bit [9]**
External abort Type. See the description of HSR.EA for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

**Bits [8:6]**
Reserved, RES0.
**DFSC, bits [5:0]**

Fault Status Code. See the description of HSR.DFSC for an SError interrupt.

This field resets to an architecturally **UNKNOWN** value.

**When the ESB instruction is executed at EL0 or EL1 and where TTBCR.EAE == 0:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>RES0</td>
<td>AET</td>
<td>RES0</td>
<td>ExT</td>
<td>RES0</td>
<td>FS[4]</td>
<td>LPAE</td>
<td>RES0</td>
<td>FS[3:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**A, bit [31]**

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

This field resets to an architecturally **UNKNOWN** value.

**Bits [30:16]**

Reserved, RES0.

**AET, bits [15:14]**

Asynchronous Error Type. See the description of DFSR.AET for an SError interrupt.

This field resets to an architecturally **UNKNOWN** value.

**Bit [13]**

Reserved, RES0.

**ExT, bit [12]**

External abort Type. See the description of DFSR.ExT for an SError interrupt.

This field resets to an architecturally **UNKNOWN** value.

**Bit [11]**

Reserved, RES0.

**FS[4], bit [10]**

This field is bit[4] of FS[4:0].

Fault Status Code. See the description of DFSR.FS for an SError interrupt.

The FS field is split as follows:

- FS[4] is DISR[10].
- FS[3:0] is DISR[3:0].

This field resets to an architecturally **UNKNOWN** value.

**LPAE, bit [9]**

Format.

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table format.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally **UNKNOWN** value.

**Bits [8:4]**

Reserved, RES0.

**FS[3:0], bits [3:0]**

This field is bits[3:0] of FS[4:0].


**When the ESB instruction is executed at EL0 or EL1 and where TTBCR.EAE == 1:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| A  | RES0 | AET | RES0 | ExT | RES0 | LPAE | RES0 | STATUS |

**A, bit [31]**

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

This field resets to an architecturally **UNKNOWN** value.

**Bits [30:16]**

Reserved, RES0.

**AET, bits [15:14]**

Asynchronous Error Type. See the description of **DFSR.AET** for an SError interrupt.

This field resets to an architecturally **UNKNOWN** value.

**Bit [13]**

Reserved, RES0.

**ExT, bit [12]**

External abort Type. See the description of **DFSR.ExT** for an SError interrupt.

This field resets to an architecturally **UNKNOWN** value.

**Bits [11:10]**

Reserved, RES0.

**LPAE, bit [9]**

Format.

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [8:6]**

Reserved, RES0.
STATUS, bits [5:0]

Fault Status Code. See the description of DFSR, DFSC for an SError interrupt.

This field resets to an architecturally UNKNOWN value.

Accessing the DISR

An indirect write to DISR made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR occurring in program order after the ESB instruction.

DISR is RAZ/WI if EL3 is implemented, the PE is in Non-debug state, and any of the following apply:

- EL3 is using AArch64, SCR_EL3.EA == 1, and any of the following apply:
  - The PE is executing at EL2.
  - The PE is executing at EL1 and ((SCR_EL3.NS == 0 && SCR_EL3.EEL2 == 0) || HCR_EL2.AMO == 0).
- EL3 is using AArch32, SCR.EA == 1, and any of the following apply:
  - The PE is executing at EL2.
  - The PE is executing at EL1 and (SCR.NS == 0 || HCR.AMO == 0).

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsid PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
    return VDISR_EL2;
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
    return VDISR;
  else
    return DISR;
else
  return DISR;
elsid PSTATE.EL == EL2 then
  return DISR;
elsid PSTATE.EL == EL3 then
  return DISR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        VDISR_EL2 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
        VDISR = R[t];
    else
        DISR = R[t];
elsif PSTATE.EL == EL2 then
    DISR = R[t];
elsif PSTATE.EL == EL3 then
    DISR = R[t];
DLR, Debug Link Register

The DLR characteristics are:

**Purpose**

In Debug state, holds the address to restart from.

**Configuration**

AArch32 System register DLR bits [31:0] are architecturally mapped to AArch64 System register DLR_EL0[31:0].

**Attributes**

DLR is a 32-bit register.

**Field descriptions**

The DLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Restart address.

**Accessing the DLR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \ <\text{coproc}>, \{#<opc1>, <Rt>, <CRn>, <CRm}\{, \{#<opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if !Halted() then
  UNDEFINED;
else
  return DLR;

\[
\text{MCR}\{<c>\}{<q>} \ <\text{coproc}>, \{#<opc1>, <Rt>, <CRn>, <CRm}\{, \{#<opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if !Halted() then
  UNDEFINED;
else
  DLR = R[t];
DSPSR, Debug Saved Program Status Register

The DSPSR characteristics are:

**Purpose**

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

**Configuration**

AArch32 System register DSPSR bits [31:0] are architecturally mapped to AArch64 System register DSPSR_EL0[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

DSPSR is a 32-bit register.

**Field descriptions**

The DSPSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| N  | Z  | C  | V  | Q  | IT[1:0] | DIT | SSBS | PAN | SS | IL | GE | IT[7:2] | E | A | I | F | T | M[4:0] |

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.
Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on entering Debug state, and copied to PSTATE.Q on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on entering Debug state, and copied to PSTATE.IT[1:0] on exiting Debug state.

On exiting Debug state DSPSR.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**DIT, bit [24]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**SSBS, bit [23]**

*When ARMv8.0-SSBS is implemented:*

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

*When ARMv8.1-PAN is implemented:*

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**SS, bit [21]**

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.
IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on entering Debug state, and copied to PSTATE.GE on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on entering Debug state, and copied to PSTATE.IT[7:2] on exiting Debug state.

DSPSR.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on entering Debug state, and copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR.E is **RES0**. If the implementation does not support little-endian operation, DSPSR.E is **RES1**. On exiting Debug state, if the implementation does not support big-endian operation at the Exception level being returned to, DSPSR.E is **RES0**, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR.E is **RES1**.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on entering Debug state, and copied to PSTATE.T on exiting Debug state.

This field resets to an architecturally **UNKNOWN** value.
**M[4:0], bits [4:0]**

Mode. Set to the value of PSTATE.M[4:0] on entering Debug state, and copied to PSTATE.M[4:0] on exiting Debug state.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10110</td>
<td>Monitor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11010</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If DSPSR.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**Accessing the DSPSR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \langle\text{coproc}\rangle, \{\#\langle\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle\}\{, \{\#\langle\text{opc2}\rangle}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if !Halted() then
  UNDEFINED;
else
  return DSPSR;

\[
\text{MCR}\{<c>\}{<q>} \langle\text{coproc}\rangle, \{\#\langle\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle\}\{, \{\#\langle\text{opc2}\rangle}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if !Halted() then
  UNDEFINED;
else
  DSPSR = R[t];

---

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbf6c3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The DTLBIALL characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from data TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

**Configuration**

**Attributes**

DTLBIALL is a 32-bit System instruction.

**Field descriptions**

DTLBIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the DTLBIALL instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>\} \text{<coproc>, } \{\#}\text{<opc1>, } <\text{Rt}, >, <\text{CRn}, >, <\text{CRm}\{, \{\#}\text{<opc2}>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DTLBIALL();
    endif
elsif PSTATE.EL == EL2 then
    DTLBIALL();
elsif PSTATE.EL == EL3 then
    DTLBIALL();
endif
DTLBIASID, Data TLB Invalidate by ASID match

The DTLBIASID characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

**Configuration**

**Attributes**

DTLBIASID is a 32-bit System instruction.

**Field descriptions**

The DTLBIASID input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 | ASID |

Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

**Executing the DTLBIASID instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE_EL == EL0 then
    UNDEFINED;
elsif PSTATE_EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        DTLBIASID(R[t]);
    end if;
elsif PSTATE_EL == EL2 then
    DTLBIASID(R[t]);
elsif PSTATE_EL == EL3 then
    DTLBIASID(R[t]);
DTLBIMVA, Data TLB Invalidate by VA

The DTLBIMVA characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

**Configuration**

**Attributes**

DTLBIMVA is a 32-bit System instruction.

**Field descriptions**

The DTLBIMVA input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA, Virtual address to match</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:8]**

Reserved, RES0.

**ASID, bits [7:0]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
Executing the DTLBIMVA instruction

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    DTLBIMVA(R[t]);
  endif
elsif PSTATE.EL == EL2 then
  DTLBIMVA(R[t]);
elsif PSTATE.EL == EL3 then
  DTLBIMVA(R[t]);

The DVRCTX characteristics are:

**Purpose**

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

When this instruction is complete and synchronized, data value prediction does not permit later speculative execution within the target execution context to be observable through side channels.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

**Note**

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

**Configuration**

This instruction is present only when ARMv8.0-PredInv is implemented. Otherwise, direct accesses to DVRCTX are UNDEFINED.

**Attributes**

DVRCTX is a 32-bit System instruction.

**Field descriptions**

The DVRCTX input value bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>GVMID NS</td>
<td>EL</td>
<td>VMID</td>
<td>RES0</td>
<td>GASID</td>
<td>ASID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**GVMID, bit [27]**

Execution of this instruction applies to all VMIDs or a specified VMID.

<table>
<thead>
<tr>
<th>GVMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified VMID for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all VMIDs for an EL0 or EL1 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>
If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

**NS, bit [26]**

Security State.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state.</td>
</tr>
</tbody>
</table>

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

**EL, bits [25:24]**

Exception Level

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>EL0.</td>
</tr>
<tr>
<td>0b01</td>
<td>EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>EL3.</td>
</tr>
</tbody>
</table>

If the instruction is executed at an exception level lower than the specified level, this instruction is treated as a NOP.

**VMID, bits [23:16]**

Only applies when bit[27] is 0 and either:
- an EL1 context.
- an EL0 context when (HCR_EL2.E2H==0 or HCR_EL2.TGE==0) or EL2 is using AArch32 state.

Otherwise this field is RES0.

When the instruction is executed at EL1 then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==0 or HCR_EL2.TGE==0 or ELUsingAArch32(EL2)) then this field is treated as the current VMID.

When the instruction is executed at EL0 and (HCR_EL2.E2H==1 and HCR_EL2.TGE==1 and !ELUsingAArch32(EL2)) then this field is ignored.

**Bits [15:9]**

Reserved, RES0.

**GASID, bit [8]**

Execution of this instruction applies to all ASIDs or a specified ASID.

<table>
<thead>
<tr>
<th>GASID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Applies to specified ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Applies to all ASID for an EL0 context. For all other contexts this field is RES0.</td>
</tr>
</tbody>
</table>

If the instruction is executed at EL0, then this field has an Effective value of 0.

**ASID, bits [7:0]**

Only applies for an EL0 context and when bit[8] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0 then this field is treated as the current ASID.
Executing the DVPRCTX instruction

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}\{<q>\} \text{<coproc>}, \{#\} \text{<opc1>}, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{#\} \text{<opc2>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() && SCTLR.EnRCTX == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    DVPRCTX(R[t]);
  endif
else
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x03);
  else
    DVPRCTX(R[t]);
  endif
endif
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ELR_hyp, Exception Link Register (Hyp mode)

The ELR_hyp characteristics are:

**Purpose**

When taking an exception to Hyp mode, holds the address to return to.

**Configuration**

AArch32 System register ELR_hyp bits [31:0] are architecturally mapped to AArch64 System register ELR_EL2[31:0]. On a reset into an Exception level that is using AArch32 ELR_hyp is **UNKNOWN**.

**Attributes**

ELR_hyp is a 32-bit register.

**Field descriptions**

The ELR_hyp bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:0]**

Return address.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ELR_hyp**

ELR_hyp is accessible only at Hyp mode and Monitor mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

**MRS{<c>}{<q>} <Rd>, ELR_hyp**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b1</td>
<td>0b1110</td>
</tr>
</tbody>
</table>

**MSR{<c>}{<q>} ELR_hyp, <Rn>**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b1</td>
<td>0b1110</td>
</tr>
</tbody>
</table>
The ERRIDR characteristics are:

**Purpose**

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

**Configuration**

AArch32 System register ERRIDR bits [31:0] are architecturally mapped to AArch64 System register `ERRIDR_EL1[31:0]`.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRIDR are `UNDEFINED`.

**Attributes**

ERRIDR is a 32-bit register.

**Field descriptions**

The ERRIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15-0</td>
<td>NUM, highest numbered index of the records plus one. Zero indicates no records can be accessed through the Error Record System registers. Each implemented record is owned by a node. A node might own multiple records.</td>
</tr>
</tbody>
</table>

**Accessing the ERRIDR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERRIDR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERRIDR;
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERRIDR;
ERRSELR, Error Record Select Register

The ERRSELR characteristics are:

**Purpose**

Selects an error record to be accessed through the Error Record System registers.

**Configuration**

AArch32 System register ERRSELR bits [31:0] are architecturally mapped to AArch64 System register `ERRSELR_EL1[31:0]`.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRSELR are **UNDEFINED**.

If `ERRIDR` indicates that zero error records are implemented, then it is **IMPLEMENTATION DEFINED** whether ERRSELR is **UNDEFINED** or `RES0`.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ERRSELR is a 32-bit register.

**Field descriptions**

The ERRSELR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>SEL</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, `RES0`.

**SEL, bits [15:0]**

Selects the error record accessed through the ERX registers.

If ERRSELR.SEL is set to a value greater than or equal to `ERRIDR.NUM`, then all of the following apply:

- The value read back from ERRSELR.SEL is **UNKNOWN**.
- One of the following occurs:
  - An **UNKNOWN** error record is selected.
  - The ERX* registers are RAZ/WI.
  - ERX* register reads and writes are NOPs.
  - ERX* register reads and writes are **UNDEFINED**.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ERRSELR**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ERRSELR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ERRSELR;
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ERRSELR;

ERRSELR, Error Record Select Register
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERRSELR = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERRSELR = R[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERRSELR = R[t];
ERXADDR, Selected Error Record Address Register

The ERXADDR characteristics are:

**Purpose**

Accesses bits [31:0] of ERR<\(n\)>ADDR for the error record selected by ERRSELR SEL.

**Configuration**

AArch32 System register ERXADDR bits [31:0] are architecturally mapped to AArch64 System register ERXADDR_EL1[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXADDR are undefined.

**Attributes**

ERXADDR is a 32-bit register.

**Field descriptions**

The ERXADDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Bits [31:0] of ERR<\(n\)>ADDR.

**Bits [31:0]**

ERXADDR accesses bits [31:0] of ERR<\(n\)>ADDR, where \(n\) is the value in ERRSELR SEL.

**Accessing the ERXADDR**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An unknown record is selected.
- ERXADDR is RAZ/WI.
- Direct reads and writes of ERXADDR are NOPs.
- Direct reads and writes of ERXADDR are undefined.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXADDR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXADDR;
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXADDR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elseif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXADDR = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXADDR = R[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXADDR = R[t];
ERXADDR2, Selected Error Record Address Register 2

The ERXADDR2 characteristics are:

**Purpose**

Accesses bits [63:32] of ERR<n>ADDR for the error record selected by ERRSEL.R SEL.

**Configuration**

AArch32 System register ERXADDR2 bits [31:0] are architecturally mapped to AArch64 System register ERXADDR_EL1[63:32].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXADDR2 are UNDEFINED.

**Attributes**

ERXADDR2 is a 32-bit register.

**Field descriptions**

The ERXADDR2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Bits [63:32] of ERR<n>ADDR. |

**Bits [31:0]**

ERXADDR2 accesses bits [63:32] of ERR<n>ADDR, where n is the value in ERRSEL.R SEL.

**Accessing the ERXADDR2**

If ERRIDR.NUM == 0 or ERRSEL.R SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXADDR2 is RAZ/WI.
- Direct reads and writes of ERXADDR2 are NOPs.
- Direct reads and writes of ERXADDR2 are UNDEFINED.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXADDR2;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
    endif
else
    ERXADDR2 = R[t];
end if

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b101</td>
<td>0b0100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
    endif
else
    ERXADDR2 = R[t];
end if
ERXCTLR, Selected Error Record Control Register

The ERXCTLR characteristics are:

**Purpose**

Accesses bits [31:0] of $\text{ERR}<n>\text{CTRL}$ for the error record selected by $\text{ERRSELR}.\text{SEL}$.

**Configuration**

AArch32 System register ERXCTLR bits [31:0] are architecturally mapped to AArch64 System register $\text{ERXCTLR}_\text{EL1}[31:0]$.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXCTLR are UNDEFINED.

**Attributes**

ERXCTLR is a 32-bit register.

**Field descriptions**

The ERXCTLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Bits [31:0] of ERR<n>CTRL | |

**Bits [31:0]**

ERXCTLR accesses bits [31:0] of $\text{ERR}<n>\text{CTRL}$, where $n$ is the value in $\text{ERRSELR}.\text{SEL}$.

**Accessing the ERXCTLR**

If $\text{ERRIDR}.\text{NUM} == 0$ or $\text{ERRSELR}.\text{SEL}$ is set to a value greater than or equal to $\text{ERRIDR}.\text{NUM}$, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXCTLR is RAZ/WI.
- Direct reads and writes of ERXCTLR are NOPs.
- Direct reads and writes of ERXCTLR are UNDEFINED.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXCTLR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];
else
    return ERXCTLR;

ERXCTLR, Selected Error Record Control Register

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXCTLR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];
ERXCTRL2, Selected Error Record Control Register 2

The ERXCTRL2 characteristics are:

**Purpose**

Accesses bits [63:32] of `ERR<n>CTRL` for the error record selected by `ERRSELR.SEL`.

**Configuration**

AArch32 System register ERXCTRL2 bits [31:0] are architecturally mapped to AArch64 System register `ERXCTRL_EL1[63:32]`.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXCTRL2 are **UNDEFINED**.

**Attributes**

ERXCTRL2 is a 32-bit register.

**Field descriptions**

The ERXCTRL2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Bits [63:32] of ERR<n>CTRL

**Accessing the ERXCTRL2**

If `ERRIDR.NUM == 0` or `ERRSELR.SEL` is set to a value greater than or equal to `ERRIDR.NUM`, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXCTRL2 is RAZ/WI.
- Direct reads and writes of ERXCTRL2 are NOPs.
- Direct reads and writes of ERXCTRL2 are **UNDEFINED**.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#<opc1>}, <Rt>, <CRn>, <CRm>{, {#<opc2>}}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  end if;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else if EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else
    return ERXCTLR2;
  end if;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch32.EL3TakeHypTrapException(0x03);
  else if EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.EL3TakeHypTrapException(0x03);
  else
    return ERXCTLR2;
  end if;
else
  return ERXCTLR2;
end if;

MCR{<coproc>}{<opc1>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else
    return ERXCTLR2;
  end if;
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else
    return ERXCTLR2;
  end if;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  end if;
elsif PSTATE.EL == EL3 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.EL2TakeHypTrapException(0x03);
  else
    return ERXCTLR2;
  end if;
else
  return ERXCTLR2;
end if;
ERXFR, Selected Error Record Feature Register

The ERXFR characteristics are:

**Purpose**

Accesses bits [31:0] of \( \text{ERR}<n>\text{FR} \) for the error record selected by \( \text{ERRSELR}.\text{SEL} \).

**Configuration**

AArch32 System register ERXFR bits [31:0] are architecturally mapped to AArch64 System register \( \text{ERXFR_EL1}[31:0] \).

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXFR are \text{UNDEFINED}.

**Attributes**

ERXFR is a 32-bit register.

**Field descriptions**

The ERXFR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Bits [31:0] of ERR\(<n>\text{FR} \)

**Bits [31:0]**

ERXFR accesses bits [31:0] of \( \text{ERR}<n>\text{FR} \), where \( n \) is the value in \( \text{ERRSELR}.\text{SEL} \).

**Accessing the ERXFR**

If \( \text{ERRIDR}.\text{NUM} == 0 \) or \( \text{ERRSELR}.\text{SEL} \) is set to a value greater than or equal to \( \text{ERRIDR}.\text{NUM} \), then one of the following occurs:

- An \text{UNKNOWN} record is selected.
- ERXFR is RAZ/WI.
- Direct reads of ERXFR are NOPs.
- Direct reads of ERXFR are \text{UNDEFINED}.

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>{<q}> \<coproc>, \{#\<opc1>, \<Rt>, \<CRn>, \<CRm>{, \{#\<opc2>\}}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
else
    return ERXFR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
else
    return ERXFR;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
else
    return ERXFR;
ERXFR2, Selected Error Record Feature Register 2

The ERXFR2 characteristics are:

Purpose

Accesses bits [63:32] of ERR<n>FR for the error record selected by ERRSELR.SEL.

Configuration

AArch32 System register ERXFR2 bits [31:0] are architecturally mapped to AArch64 System register ERXFR_EL1[63:32].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXFR2 are UNDEFINED.

Attributes

ERXFR2 is a 32-bit register.

Field descriptions

The ERXFR2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:32] of ERR<n>FR

Accessing the ERXFR2

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXFR2 is RAZ/WI.
- Direct reads of ERXFR2 are NOPs.
- Direct reads of ERXFR2 are UNDEFINED.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() &&!ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) &&!ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXFR2;
else PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXFR2;
else PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXFR2;

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The ERXMISC0 characteristics are:

**Purpose**

Accesses bits [31:0] of ERR<n>MISC0 for the error record selected by ERRSELR.SEL.

**Configuration**

AArch32 System register ERXMISC0 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC0_EL1[31:0].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXMISC0 are UNDEFINED.

**Attributes**

ERXMISC0 is a 32-bit register.

**Field descriptions**

The ERXMISC0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:0]**

ERXMISC0 accesses bits [31:0] of ERR<n>MISC0, where n is the value in ERRSELR.SEL.

**Accessing the ERXMISC0**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC0 is RAZ/WI.
- Direct reads and writes of ERXMISC0 are NOPs.
- Direct reads and writes of ERXMISC0 are UNDEFINED.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ERXMISC0;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC0 = R[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ERXMISC0;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC0 = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC0 = R[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC0 = R[t];
The ERXMISC1 characteristics are:

**Purpose**

Accesses bits [63:32] of ERR<n>MISC0 for the error record selected by ERRSELR.SEL.

**Configuration**

AArch32 System register ERXMISC1 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC0_EL1[63:32].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXMISC1 are UNDEFINED.

**Attributes**

ERXMISC1 is a 32-bit register.

**Field descriptions**

The ERXMISC1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Bits [63:32] of ERR&lt;n&gt;MISC0</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

ERXMISC1 accesses bits [63:32] of ERR<n>MISC0, where n is the value in ERRSELR.SEL.

**Accessing the ERXMISC1**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC1 is RAZ/WI.
- Direct reads and writes of ERXMISC1 are NOPs.
- Direct reads and writes of ERXMISC1 are UNDEFINED.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC1;
    elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC1 = R[t];
The ERXMISC2 characteristics are:

**Purpose**

Accesses bits [31:0] of \texttt{ERR<n>MISC1} for the error record selected by \texttt{ERRSELR}.SEL.

**Configuration**

AArch32 System register ERXMISC2 bits [31:0] are architecturally mapped to AArch64 System register \texttt{ERXMISC1\_EL1[31:0]}.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXMISC2 are UNDEFINED.

**Attributes**

ERXMISC2 is a 32-bit register.

**Field descriptions**

The ERXMISC2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Bits [31:0] of ERR<n>MISC1 |

**Bits [31:0]**

ERXMISC2 accesses bits [31:0] of \texttt{ERR<n>MISC1}, where \texttt{n} is the value in \texttt{ERRSELR}.SEL.

**Accessing the ERXMISC2**

If \texttt{ERRIDR\_NUM} == 0 or \texttt{ERRSELR}.SEL is set to a value greater than or equal to \texttt{ERRIDR}.NUM, then one of the following occurs:

- An \texttt{UNKNOWN} record is selected.
- ERXMISC2 is RAZ/WI.
- Direct reads and writes of ERXMISC2 are NOPs.
- Direct reads and writes of ERXMISC2 are UNDEFINED.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\}\ <\text{coproc}>\{<opcl>\}, \ (#)\<\text{opcm}\}, \ <\text{Rt}\}, \ <\text{CRn}\}, \ <\text{CRm}\}, \ (#)\<\text{opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsf EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsf EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsf HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXMISC2;
elsf PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsf HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXMISC2;
elsf PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXMISC2;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsf EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsf EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsf EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
elsf HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsf HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC2 = R[t];
elsf PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsf HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC2 = R[t];
elsf PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC2 = R[t];
ERXMISC3, Selected Error Record Miscellaneous Register 3

The ERXMISC3 characteristics are:

**Purpose**

Accesses bits [63:32] of ERR<n>MISC1 for the error record selected by ERRSELR.SEL.

**Configuration**

AArch32 System register ERXMISC3 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC1_EL1[63:32].

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXMISC3 are UNDEFINED.

**Attributes**

ERXMISC3 is a 32-bit register.

**Field descriptions**

The ERXMISC3 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits [63:32] of ERR&lt;n&gt;MISC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

ERXMISC3 accesses bits [63:32] of ERR<n>MISC1, where n is the value in ERRSELR.SEL.

**Accessing the ERXMISC3**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An UNKNOWN record is selected.
- ERXMISC3 is RAZ/WI.
- Direct reads and writes of ERXMISC3 are NOPs.
- Direct reads and writes of ERXMISC3 are UNDEFINED.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == ’1’ then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == ’1’ then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == ’1’ then
    AArch32.TakeMonitorTrapException();
  else
    return ERXMISC3;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == ’1’ then
    AArch32.TakeMonitorTrapException();
  else
    return ERXMISC3;
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == ’1’ then
    AArch32.TakeMonitorTrapException();
  else
    return ERXMISC3;
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == ’1’ then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == ’1’ then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == ’1’ then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC3 = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == ’1’ then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == ’1’ then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC3 = R[t];
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == ’1’ then
    AArch32.TakeMonitorTrapException();
  else
    ERXMISC3 = R[t];
ERXMISC4, Selected Error Record Miscellaneous Register 4

The ERXMISC4 characteristics are:

**Purpose**

Accesses bits [31:0] of \texttt{ERR<n>MISC2} for the error record selected by \texttt{ERRSELR}.SEL.

**Configuration**

AArch32 System register ERXMISC4 bits [31:0] are architecturally mapped to AArch64 System register \texttt{ERXMISC2_EL1[31:0]}.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXMISC4 are UNDEFINED.

**Attributes**

ERXMISC4 is a 32-bit register.

**Field descriptions**

The ERXMISC4 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Bits [31:0] of \texttt{ERR<n>MISC2}

**Accessing the ERXMISC4**

If \texttt{ERRIDR}.NUM == 0 or \texttt{ERRSELR}.SEL is set to a value greater than or equal to \texttt{ERRIDR}.NUM, then one of the following occurs:

- An \texttt{UNKNOWN} record is selected.
- ERXMISC4 is RAZ/WI.
- Direct reads and writes of ERXMISC4 are NOPs.
- Direct reads and writes of ERXMISC4 are UNDEFINED.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \ <\text{coproc}> , \ <\text{opc1}> , \ <\text{CRn}> , \ <\text{CRm}> , \ <\#>\text{<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXMISC4;
endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC4 = R[t];
endif
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC4 = R[t];
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  return ERXMISC4;
endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC4 = R[t];
endif
elsif PSTATE.EL == EL3 then
  if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
    AArch32.TakeMonitorTrapException();
else
  ERXMISC4 = R[t];
endif
The ERXMISC5 characteristics are:

**Purpose**

Accesses bits [63:32] of ERR\textless{}n\textgreater{}MISC2 for the error record selected by ERRSELR.SEL.

**Configuration**

AArch32 System register ERXMISC5 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC2_EL1[63:32].

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXMISC5 are UNDEFINED.

**Attributes**

ERXMISC5 is a 32-bit register.

**Field descriptions**

The ERXMISC5 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Bits [63:32] of ERR\textless{}n\textgreater{}MISC2 |

**Bits [31:0]**

ERXMISC5 accesses bits [63:32] of ERR\textless{}n\textgreater{}MISC2, where n is the value in ERRSELR.SEL.

**Accessing the ERXMISC5**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC5 is RAZ/WI.
- Direct reads and writes of ERXMISC5 are NOPs.
- Direct reads and writes of ERXMISC5 are UNDEFINED.

Accesses to this register use the following encodings:

\[ \text{MRC}\{\text{<c>}\}\{\text{<q>}\} \text{ <coproc>, \{#\<opc1>, \<Rt>, \<CRn>, \<CRm>\} \{\#\<opc2>\} } \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeHypTrapException();
    else
        return ERXMISC5;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC5;
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC5;
    endif
else
    ERXMISC5 = R[t];
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC5 = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC5 = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC5 = R[t];
    endif
ERXMISC6, Selected Error Record Miscellaneous Register 6

The ERXMISC6 characteristics are:

**Purpose**

Accesses bits [31:0] of ERR<n>MISC3 for the error record selected by ERRSELR.SEL.

**Configuration**

AArch32 System register ERXMISC6 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC3_EL1[31:0].

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXMISC6 are UNDEFINED.

**Attributes**

ERXMISC6 is a 32-bit register.

**Field descriptions**

The ERXMISC6 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>30</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>29</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>28</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>27</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>26</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>25</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>24</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>23</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>22</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>21</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>20</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>19</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>18</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>17</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>16</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>15</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>14</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>13</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>12</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>11</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>10</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>9</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>8</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>7</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>6</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>5</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>4</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>3</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>2</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>1</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
<tr>
<td>0</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

ERXMISC6 accesses bits [31:0] of ERR<n>MISC3, where n is the value in ERRSELR.SEL.

**Accessing the ERXMISC6**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC6 is RAZ/WI.
- Direct reads and writes of ERXMISC6 are NOPs.
- Direct reads and writes of ERXMISC6 are **UNDEFINED**.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException(0x03);
    else
        return ERXMISC6;
    endif;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC6;
    endif;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC6;
    endif;
endif;

MCR{<c>}{<q>} {<coproc>}, {#}<opc1>, <Rt>, <CRn>, {<CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC6 = R[t];
    endwhile;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC6 = R[t];
    endwhile;
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC6 = R[t];
    endwhile;
The ERXMISC7 characteristics are:

**Purpose**

Accesses bits [63:32] of ERR<n>MISC3 for the error record selected by ERRSELR.SEL.

**Configuration**

AArch32 System register ERXMISC7 bits [31:0] are architecturally mapped to AArch64 System register ERXMISC3_EL1[63:32].

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERXMISC7 are UNDEFINED.

**Attributes**

ERXMISC7 is a 32-bit register.

**Field descriptions**

The ERXMISC7 bit assignments are:

```
<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

**Bits [31:0]**

ERXMISC7 accesses bits [63:32] of ERR<n>MISC3, where n is the value in ERRSELR.SEL.

**Accessing the ERXMISC7**

If ERRIDR.NUM == 0 or ERRSELR.SEL is set to a value greater than or equal to ERRIDR.NUM, then one of the following occurs:

- An **UNKNOWN** record is selected.
- ERXMISC7 is RAZ/WI.
- Direct reads and writes of ERXMISC7 are NOPs.
- Direct reads and writes of ERXMISC7 are **UNDEFINED**.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC7;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXMISC7;
    endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXMISC7 = R[t];
    endif
ERXSTATUS, Selected Error Record Primary Status Register

The ERXSTATUS characteristics are:

**Purpose**

Accesses bits [31:0] of `ERR<n>STATUS` for the error record selected by `ERRSELr_SEL`.

**Configuration**

AArch32 System register ERXSTATUS bits [31:0] are architecturally mapped to AArch64 System register `ERXSTATUS_EL1[31:0]`.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERXSTATUS are `UNDEFINED`.

**Attributes**

ERXSTATUS is a 32-bit register.

**Field descriptions**

The ERXSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits [31:0] of ERR&lt;n&gt;STATUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

ERXSTATUS accesses bits [31:0] of `ERR<n>STATUS`, where n is the value in `ERRSELr_SEL`.

**Accessing the ERXSTATUS**

If `ERRIDr_NUM == 0` or `ERRSELr_SEL` is set to a value greater than or equal to `ERRIDr_NUM`, then one of the following occurs:

- An `UNKNOWN` record is selected.
- ERXSTATUS is RAZ/WI.
- Direct reads and writes of ERXSTATUS are NOPs.
- Direct reads and writes of ERXSTATUS are `UNDEFINED`.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#<opc1>, <Rt>, <CRn>, <CRm>{, {#<opc2>}}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ERXSTATUS;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXSTATUS = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXSTATUS = R[t];
FCSEIDR, FCSE Process ID register

The FCSEIDR characteristics are:

**Purpose**

Identifies whether the Fast Context Switch Extension (FCSE) is implemented.

From Armv8, the FCSE is not implemented, so this register is RAZ/WI. Software can access this register to determine that the implementation does not include the FCSE.

**Configuration**

**Attributes**

FCSEIDR is a 32-bit register.

**Field descriptions**

The FCSEIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAZ/WI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RAZ/WI.

**Accessing the FCSEIDR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return FCSEIDR;
    endif
elsif PSTATE.EL == EL2 then
    return FCSEIDR;
elsif PSTATE.EL == EL3 then
    return FCSEIDR;
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>

Page 2439
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        FCSEIDR = R[t];
    end if;
elsif PSTATE.EL == EL2 then
    FCSEIDR = R[t];
elsif PSTATE.EL == EL3 then
    FCSEIDR = R[t];

FPEXC, Floating-Point Exception Control register

The FPEXC characteristics are:

**Purpose**

Provides a global enable for the implemented Advanced SIMD and floating-point functionality, and reports floating-point status information.

**Configuration**

AArch32 System register FPEXC bits [31:0] are architecturally mapped to AArch64 System register FPEXC32_EL2[31:0].

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

FPEXC is a 32-bit register.

**Field descriptions**

The FPEXC bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>EX</td>
<td>Exception bit. From Armv8, this bit is RAZ/WI.</td>
</tr>
<tr>
<td></td>
<td>EN</td>
<td>Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:</td>
</tr>
<tr>
<td>0</td>
<td>0b0</td>
<td>VMSR accesses to the [<strong>FPEXC</strong> or <strong>FPSID</strong>.]</td>
</tr>
<tr>
<td></td>
<td>0b1</td>
<td>VMRS accesses from the [<strong>FPEXC</strong>, <strong>FPSID</strong>, <strong>MVFR0</strong>, <strong>MVFR1</strong>, or <strong>MVFR2</strong>.]</td>
</tr>
</tbody>
</table>

Execution of floating-point and Advanced SIMD instructions in AArch32 state can be disabled or trapped by the following controls:

- **CPACR**, cp10, or, if executing at EL0, **CPACR_EL1.FPEN**.
- **FPEXC.EN**.
- If executing in Non-secure state:
  - **HCTR**, TCP10, or if EL2 is using AArch64, **CPR_EL2.TFP**.
  - **NSACR**, cp10, or if EL3 is using AArch64, **CPR_EL3.TFP**.
- For Advanced SIMD instructions only:
If executing in Non-secure state, HCPTR.TASE and NSACR.NSTRCDIS.

See the descriptions of the controls for more information.

**Note**

When executing at EL0 using AArch32:

- If EL1 is using AArch64 then behavior is as if the value of FPEXC.EN is 1.
- If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is \{1, 1\}, then the behavior is as if the value of FPEXC.EN is 1.
- If EL2 is using AArch64 and enabled in the current Security state, and the value of HCR_EL2.{RW, TGE} is \{0, 1\}, then it is IMPLEMENTATION DEFINED whether the behavior is:
  - As if the value of FPEXC.EN is 1.
  - Determined by the value of FPEXC.EN, as described in this field description. However, Arm deprecates using the value of FPEXC.EN to determine behavior.

This field resets to 0.

**DEX, bit [29]**

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC.TFV field is valid.

The meaning of this bit is:

<table>
<thead>
<tr>
<th>DEX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC.TFV is RW then it is invalid and UNKNOWN. If FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN.</td>
</tr>
<tr>
<td>0b1</td>
<td>The exception was generated during the execution of an unallocated encoding. FPEXC.TFV is valid and indicates the cause of the exception.</td>
</tr>
</tbody>
</table>

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the FPSCR.{Stride, Len} fields as RAZ, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

**FP2V, bit [28]**

FPINST2 instruction valid bit. From Armv8, this bit is RES0.

This field resets to an architecturally UNKNOWN value.

**VV, bit [27]**

VECITR valid bit. From Armv8, this bit is RES0.

This field resets to an architecturally UNKNOWN value.
TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC.DEX is 1. When valid, it indicates the cause of the exception and therefore whether the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are valid.

<table>
<thead>
<tr>
<th>TFV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMLU, VSDIV, VSQRT, or VSUB instruction when one or both of FPCR, {Stride, Len} was non-zero. If the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are RW then they are invalid and UNKNOWN.</td>
</tr>
<tr>
<td>0b1</td>
<td>FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception.</td>
</tr>
</tbody>
</table>

This bit returns a status value and ignores writes.

When the value of FPEXC.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

On an implementation that supports the trapping of floating-point exceptions and implements FPCR, {Stride, Len} as RAZ, this bit is RAO/WI.

This field resets to an architecturally UNKNOWN value.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector iteration count. From Armv8, this field is RES1.

This field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while FPCR, IDE was 1:

<table>
<thead>
<tr>
<th>IDF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input Denormal exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input Denormal exception has occurred.</td>
</tr>
</tbody>
</table>

Input Denormal exceptions can occur only when FPCR, FZ is 1.

Note

A half-precision floating-point value that is flushed to zero because the value of FPCR, FZ16 is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.
IXF, bit [4]

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while `FPSCR.IXE` was 1:

<table>
<thead>
<tr>
<th>IXF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Inexact exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inexact exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

UFF, bit [3]

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while `FPSCR.UFE` was 1:

<table>
<thead>
<tr>
<th>UFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow exception has occurred.</td>
</tr>
</tbody>
</table>

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when `FPSCR.FZ16` is 0.
- Otherwise only when `FPSCR.FZ` is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

OFF, bit [2]

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while `FPSCR.OFE` was 1:

<table>
<thead>
<tr>
<th>OFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while `FPSCR.DZE` was 1:

<table>
<thead>
<tr>
<th>DZF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Divide by Zero exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Divide by Zero exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.
When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

**IOF, bit [0]**

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while **FPSCR.IOE** was 1:

<table>
<thead>
<tr>
<th>IOF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Invalid Operation exception has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invalid Operation exception has occurred.</td>
</tr>
</tbody>
</table>

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and **UNKNOWN**.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the FPEXC**

Accesses to this register use the following encodings:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

<table>
<thead>
<tr>
<th>reg</th>
<th>0b1000</th>
</tr>
</thead>
</table>

FPEXC, Floating-Point Exception Control register
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ELUsingAArch32(EL1) && (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
       CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
        NSACR.cp10 == '0') || HCPR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    else
        return FPEXC;
    endif
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
        NSACR.cp10 == '0') || HCPR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    else
        return FPEXC;
    endif
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        return FPEXC;
    endif
else
end if

VMSR{<c>}{<q>} <spec_reg>, <Rt>

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
    CPACR.cp10 == '00') then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' &&
    CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' &&
    CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x08);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x08);
  else
    FPEXC = R[t];
  elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
      NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
      AArch32.TakeHypTrapException(0x00);
    else
      FPEXC = R[t];
  elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
      UNDEFINED;
    else
      FPEXC = R[t];
  end if

FPEXC, Floating-Point Exception Control register
FPSCR, Floating-Point Status and Control Register

The FPSCR characteristics are:

**Purpose**

Provides floating-point system status information and control.

**Configuration**

The named fields in this register map to the equivalent fields in the AArch64 FPCR and FPSR.

It is IMPLEMENTATION DEFINED whether the Len and Stride fields can be programmed to non-zero values, which will cause some AArch32 floating-point instruction encodings to be UNDEFINED, or whether these fields are RAZ.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

FPSCR is a 32-bit register.

**Field descriptions**

The FPSCR bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
N | Z | V | Q | A | H | P | D | N | F | Z | R | M | o | d | e | S | t | r | i | d | e | F | Z | 1 | 6
```

**N, bit [31]**

Negative condition flag. This is updated by floating-point comparison operations.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero condition flag. This is updated by floating-point comparison operations.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry condition flag. This is updated by floating-point comparison operations.

This field resets to an architecturally UNKNOWN value.

**V, bit [28]**

Overflow condition flag. This is updated by floating-point comparison operations.

This field resets to an architecturally UNKNOWN value.
QC, bit [27]

Cumulative saturation bit, Advanced SIMD only. This bit is set to 1 to indicate that an Advanced SIMD integer
operation has saturated since 0 was last written to this bit.

This field resets to an architecturally **UNKNOWN** value.

AHP, bit [26]

Alternative half-precision control bit:

<table>
<thead>
<tr>
<th>AHP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IEEE half-precision format selected.</td>
</tr>
<tr>
<td>0b1</td>
<td>Alternative half-precision format selected.</td>
</tr>
</tbody>
</table>

This bit is only used for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the ARMv8.2-FP16 extension always use the IEEE half-precision
format, and ignore the value of this bit.

This field resets to an architecturally **UNKNOWN** value.

DN, bit [25]

Default NaN mode control bit:

<table>
<thead>
<tr>
<th>DN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>NaN operands propagate through to the output of a floating-point operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any operation involving one or more NaNs returns the Default NaN.</td>
</tr>
</tbody>
</table>

The value of this bit only controls scalar floating-point arithmetic. Advanced SIMD arithmetic always uses the Default NaN setting, regardless of the value of the DN bit.

This field resets to an architecturally **UNKNOWN** value.

FZ, bit [24]

Flush-to-zero mode control bit:

<table>
<thead>
<tr>
<th>FZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard.</td>
</tr>
<tr>
<td>0b1</td>
<td>Flush-to-zero mode enabled.</td>
</tr>
</tbody>
</table>

The value of this bit only controls scalar floating-point arithmetic. Advanced SIMD arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.

This bit has no effect on half-precision calculations.

This field resets to an architecturally **UNKNOWN** value.

RMode, bits [23:22]

Rounding Mode control field. The encoding of this field is:

<table>
<thead>
<tr>
<th>RMode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Round to Nearest (RN) mode.</td>
</tr>
<tr>
<td>0b01</td>
<td>Round towards Plus Infinity (RP) mode.</td>
</tr>
<tr>
<td>0b10</td>
<td>Round towards Minus Infinity (RM) mode.</td>
</tr>
<tr>
<td>0b11</td>
<td>Round towards Zero (RZ) mode.</td>
</tr>
</tbody>
</table>

The specified rounding mode is used by almost all scalar floating-point instructions. Advanced SIMD arithmetic always uses the Round to Nearest setting, regardless of the value of the RMode bits.
This field resets to an architecturally **UNKNOWN** value.

**Stride, bits [21:20]**

It is **IMPLEMENTATION DEFINED** whether this field is RW or RAZ.

If this field is RW and is set to a value other than zero, some floating-point instruction encodings are **UNDEFINED**. The instruction pseudocode identifies these instructions.

Arm strongly recommends that software never sets this field to a value other than zero.

The value of this field is ignored when processing Advanced SIMD instructions.

This field resets to an architecturally **UNKNOWN** value.

**FZ16, bit [19]**

*When ARMv8.2-FP16 is implemented:*

Flush-to-zero mode control bit on half-precision data-processing instructions:

<table>
<thead>
<tr>
<th>FZ16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard.</td>
</tr>
<tr>
<td>0b1</td>
<td>Flush-to-zero mode enabled.</td>
</tr>
</tbody>
</table>

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, **RES0**.

**Len, bits [18:16]**

It is **IMPLEMENTATION DEFINED** whether this field is RW or RAZ.

If this field is RW and is set to a value other than zero, some floating-point instruction encodings are **UNDEFINED**. The instruction pseudocode identifies these instructions.

Arm strongly recommends that software never sets this field to a value other than zero.

The value of this field is ignored when processing Advanced SIMD instructions.

This field resets to an architecturally **UNKNOWN** value.

**IDE, bit [15]**

Input Denormal floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>IDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the IDC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IDC bit. The trap handling software can decide whether to set the IDC bit to 1.</td>
</tr>
</tbody>
</table>

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is **RAZ/WI**.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

This field resets to an architecturally **UNKNOWN** value.
Reserved, RES0.

**IXE, bit [12]**

Inexact floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>IXE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the IXC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IXC bit. The trap handling software can decide whether to set the IXC bit to 1.</td>
</tr>
</tbody>
</table>

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

This field resets to an architecturally **UNKNOWN** value.

**UFE, bit [11]**

Underflow floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>UFE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the UFC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the UFC bit. The trap handling software can decide whether to set the UFC bit to 1.</td>
</tr>
</tbody>
</table>

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

This field resets to an architecturally **UNKNOWN** value.

**OFE, bit [10]**

Overflow floating-point exception trap enable. Possible values are:

<table>
<thead>
<tr>
<th>OFE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Untrapped exception handling selected. If the floating-point exception occurs then the OFC bit is set to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the OFC bit. The trap handling software can decide whether to set the OFC bit to 1.</td>
</tr>
</tbody>
</table>

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

This field resets to an architecturally **UNKNOWN** value.

**DZE, bit [9]**

Divide by Zero floating-point exception trap enable. Possible values are:
DZE, bit [9]

Untrapped exception handling selected. If the floating-point exception occurs then the DZC bit is set to 1.

Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the DZC bit. The trap handling software can decide whether to set the DZC bit to 1.

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

This field resets to an architecturally UNKNOWN value.

IOE, bit [8]

Invalid Operation floating-point exception trap enable. Possible values are:

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

This field resets to an architecturally UNKNOWN value.

IDC, bit [7]

Input Denormal cumulative floating-point exception bit. This bit is set to 1 to indicate that the Input Denormal floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IDE bit.

Advanced SIMD instructions set this bit if the Input Denormal floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IDE bit.

This field resets to an architecturally UNKNOWN value.

Reserved, RES0.

IXC, bit [4]

Inexact cumulative floating-point exception bit. This bit is set to 1 to indicate that the Inexact floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IXE bit.

Advanced SIMD instructions set this bit if the Inexact floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IXE bit.

This field resets to an architecturally UNKNOWN value.

UFC, bit [3]

Underflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Underflow floating-point exception has occurred since 0 was last written to this bit.
How VFP instructions update this bit depends on the value of the UFE bit.

Advanced SIMD instructions set this bit if the Underflow floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the UFE bit.

This field resets to an architecturally **UNKNOWN** value.

**OFC, bit [2]**

Overflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Overflow floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the OFE bit.

Advanced SIMD instructions set this bit if the Overflow floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the OFE bit.

This field resets to an architecturally **UNKNOWN** value.

**DZC, bit [1]**

Divide by Zero cumulative floating-point exception bit. This bit is set to 1 to indicate that the Divide by Zero floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the DZE bit.

Advanced SIMD instructions set this bit if the Divide by Zero floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the DZE bit.

This field resets to an architecturally **UNKNOWN** value.

**IOC, bit [0]**

Invalid Operation cumulative floating-point exception bit. This bit is set to 1 to indicate that the Invalid Operation floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IOE bit.

Advanced SIMD instructions set this bit if the Invalid Operation floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IOE bit.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the FPSCR**

Accesses to this register use the following encodings:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if CPACR_EL1.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL1, 0x07);
  elsif ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
    CPACR.cp10 == '0') then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x08);
  else
    return FPSCR;
  endif
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x00);
  else
    return FPSCR;
  endif
elsif PSTATE.EL == EL3 then
  if CPACR.cp10 == '00' then
    UNDEFINED;
  else
    return FPSCR;
  endif
endif

VMSR{<c>}{<q>} <spec_reg>, <Rt>

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if CPACR_EL1.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL1, 0x07);
    elsif ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00') then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
   elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x08);
else
    FPSCR = R[t];
else
    if PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        else
            FPSCR = R[t];
else
    if CPACR.cp10 == '00' then
        UNDEFINED;
else
    FPSCR = R[t];
**FPSID, Floating-Point System ID register**

The FPSID characteristics are:

**Purpose**

Provides top-level information about the floating-point implementation.

This register largely duplicates information held in the MIDR. Arm deprecates use of it.

**Configuration**

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

FPSID is a 32-bit register.

**Field descriptions**

The FPSID bit assignments are:

<table>
<thead>
<tr>
<th>Implementer</th>
<th>SW</th>
<th>Subarchitecture</th>
<th>PartNum</th>
<th>Variant</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29</td>
<td>28 27</td>
<td>26 25 24 23</td>
<td>22 21</td>
<td>20 19</td>
<td>18 17 16</td>
</tr>
</tbody>
</table>

**Implementer, bits [31:24]**

Implementer codes are the same as those used for the MIDR.

For an implementation by Arm this field is 0x41, the ASCII code for A.

This field resets to an architecturally **UNKNOWN** value.

**SW, bit [23]**

Software bit. Defined values are:

<table>
<thead>
<tr>
<th>SW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation provides a hardware implementation of the floating-point instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports only software emulation of the floating-point instructions.</td>
</tr>
</tbody>
</table>

In Armv8-A the only permitted value is 0b0.

This field resets to an architecturally **UNKNOWN** value.

**Subarchitecture, bits [22:16]**

Subarchitecture version number. For an implementation by Arm, defined values are:
**Subarchitecture** | **Meaning**  
--- | ---  
0b0000000 | VFPv1 architecture with an IMPLEMENTATION DEFINED subarchitecture.  
0b0000001 | VFPv2 architecture with Common VFP subarchitecture v1.  
0b0000010 | VFPv3 architecture, or later, with Common VFP subarchitecture v2. The VFP architecture version is indicated by the \( \text{MVFR0} \) and \( \text{MVFR1} \) registers.  
0b0000011 | VFPv3 architecture, or later, with Null subarchitecture. The entire floating-point implementation is in hardware, and no software support code is required. The VFP architecture version is indicated by the \( \text{MVFR0} \) and \( \text{MVFR1} \) registers. This value can be used only by an implementation that does not support the trap enable bits in the \( \text{FPSCR} \).  
0b0000100 | VFPv3 architecture, or later, with Common VFP subarchitecture v3, and support for trap enable bits in \( \text{FPSCR} \). The VFP architecture version is indicated by the \( \text{MVFR0} \) and \( \text{MVFR1} \) registers.  

For a subarchitecture designed by Arm the most significant bit of this field, register bit[22], is 0. Values with a most significant bit of 0 that are not listed here are reserved.

When the subarchitecture designer is not Arm, the most significant bit of this field, register bit[22], must be 1. Each implementer must maintain its own list of subarchitectures it has designed, starting at subarchitecture version number 0x40.

In Armv8-A the permitted values are 0b0000011 and 0b0000100.

This field resets to an architecturally UNKNOWN value.

**PartNum, bits [15:8]**

An IMPLEMENTATION DEFINED part number for the floating-point implementation, assigned by the implementer.

This field resets to an architecturally UNKNOWN value.

**Variant, bits [7:4]**

An IMPLEMENTATION DEFINED variant number. Typically, this field distinguishes between different production variants of a single product.

This field resets to an architecturally UNKNOWN value.

**Revision, bits [3:0]**

An IMPLEMENTATION DEFINED revision number for the floating-point implementation.

This field resets to an architecturally UNKNOWN value.

**Accessing the FPSID**

Accesses to this register use the following encodings:

\[
\text{VMRS}\{<c>\}{<q>} \ <Rt>, \ <\text{spec\_reg}> 
\]

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
</tr>
</tbody>
</table>

Page 2457
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
    CPACR.cp10 == '00') then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && (ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x08);
  else
    return FPSID;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && (ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x08);
  else
    return FPSID;
elsif PSTATE.EL == EL3 then
  if CPACR.cp10 == '00' then
    UNDEFINED;
  else
    return FPSID;

VMSR{<c>}{<q>} <spec_reg>, <Rt>

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00') then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPACR.cp10 == '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x08);
    else
        //no operation
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elseif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elseif EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x08);
        else
            //no operation
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            //no operation
    end

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
HACR, Hyp Auxiliary Configuration Register

The HACR characteristics are:

Purpose

Controls trapping to Hyp mode of IMPLEMENTATION DEFINED aspects of Non-secure EL1 or EL0 operation.

Configuration

AArch32 System register HACR bits [31:0] are architecturally mapped to AArch64 System register HACR_EL2[31:0]. If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

HACR is a 32-bit register.

Field descriptions

The HACR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

Accessing the HACR

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b111</td>
</tr>
</tbody>
</table>

Page 2460
if \( \text{PSTATE.EL} == \text{EL0} \) then
\[ \text{UNDEFINED}; \]
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
  if \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR_EL2.T1} == '1' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR.T1} == '1' \) then
    \text{AArch32.TakeHypTrapException(0x03)};
  else
    \text{UNDEFINED};
  end
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
  return \text{HACR};
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
  if \( \text{SCR.NS} == '0' \) then
    \text{UNDEFINED};
  else
    \text{HACR} = \text{R[t]};
  end
\]

MCR{<c>}{<q>} {<coproc>}, {#<opc1>}, <Rt>, <CRn>, <CRm>{, {#<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if \( \text{PSTATE.EL} == \text{EL0} \) then
\[ \text{UNDEFINED}; \]
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
  if \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR_EL2.T1} == '1' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR.T1} == '1' \) then
    \text{AArch32.TakeHypTrapException(0x03)};
  else
    \text{UNDEFINED};
  end
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
  \text{HACR} = \text{R[t]};
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
  if \( \text{SCR.NS} == '0' \) then
    \text{UNDEFINED};
  else
    \text{HACR} = \text{R[t]};
  end

HACTLR, Hyp Auxiliary Control Register

The HACTLR characteristics are:

**Purpose**

Controls IMPLEMENTATION DEFINED features of Hyp mode operation.

**Configuration**

AArch32 System register HACTLR bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HACTLR is a 32-bit register.

**Field descriptions**

The HACTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the HACTLR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HACTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HACTLR;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACTLR = R[t];
end if;
HACTLR2, Hyp Auxiliary Control Register 2

The HACTLR2 characteristics are:

**Purpose**

Provides additional space to the HACTLR register to hold *IMPLEMENTATION DEFINED* trap functionality.

**Configuration**

AArch32 System register HACTLR2 bits [31:0] are architecturally mapped to AArch64 System register ACTLR_EL2[63:32].

In Armv8.0 and Armv8.1, it is *IMPLEMENTATION DEFINED* whether this register is implemented, or whether it causes *UNDEFINED* exceptions when accessed. The implementation of this register can be detected by examining *ID_MMFR4.AC2*.

From Armv8.2 this register must be implemented.

RW fields in this register reset to architecturally *UNKNOWN* values.

**Attributes**

HACTLR2 is a 32-bit register.

**Field descriptions**

The HACTLR2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*IMPLEMENTATION DEFINED*, bits [31:0]

*IMPLEMENTATION DEFINED*.

This field resets to an architecturally *UNKNOWN* value.

**Accessing the HACTLR2**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \ <\text{coproc}, \ {#}<\text{opc}1>, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}\{, \ {#}<\text{opc}2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return HACTLR2;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    return HACTLR2;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  HACTLR2 = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HACTLR2 = R[t];
HADFSR, Hyp Auxiliary Data Fault Status Register

The HADFSR characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED syndrome information for Data Abort exceptions taken to Hyp mode.

**Configuration**

AArch32 System register HADFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR0_EL2[31:0].

This is an optional register. An implementation that does not require this register can implement it as RES0.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HADFSR is a 32-bit register.

**Field descriptions**

The HADFSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [31:0]**

This field resets to an architecturally UNKNOWN value.

**Accessing the HADFSR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HADFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HADFSR;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HADFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HADFSR = R[t];
HAIFSR, Hyp Auxiliary Instruction Fault Status Register

The HAIFSR characteristics are:

**Purpose**

Provides additional IMPLEMENTATION DEFINED syndrome information for Prefetch Abort exceptions taken to Hyp mode.

**Configuration**

AArch32 System register HAIFSR bits [31:0] are architecturally mapped to AArch64 System register AFSR1_EL2[31:0].

This is an optional register. An implementation that does not require this register can implement it as RES0.

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HAIFSR is a 32-bit register.

**Field descriptions**

The HAIFSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| IMPLEMENTATION DEFINED | IMPLEMENTATION DEFINED, bits [31:0]

**IMPLEMENATION DEFINED.**

This field resets to an architecturally UNKNOWN value.

**Accessing the HAIFSR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HAIFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HAIFSR = R[t];
    end if;
else
    HAIFSR = R[t];
end if;

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

MCR{<c>}{<q>} {<coproc>}, {#}<opc1>, <Rt>, <CRn>, {<CRm>}, {#}<opc2>
HAMAIR0, Hyp Auxiliary Memory Attribute Indirection Register 0

The HAMAIR0 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by HMAIR0. These IMPLEMENTATION DEFINED attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in HMAIR0.

**Configuration**

AArch32 System register HAMAIR0 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HAMAIR0 is a 32-bit register.

**Field descriptions**

The HAMAIR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

If an implementation does not provide any IMPLEMENTATION DEFINED memory attributes, this register is RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the HAMAIR0**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HAMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HAMAIR0;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HAMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HAMAIR0 = R[t];
    end if;
HAMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1

The HAMAIR1 characteristics are:

**Purpose**

Provides IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by HMAIR1. These IMPLEMENTATION DEFINED attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in HMAIR1.

**Configuration**

AArch32 System register HAMAIR1 bits [31:0] are architecturally mapped to AArch64 System register AMAIR_EL2[63:32].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HAMAIR1 is a 32-bit register.

**Field descriptions**

The HAMAIR1 bit assignments are:

If an implementation does not provide any IMPLEMENTATION DEFINED memory attributes, this register is RES0.

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the HAMAIR1**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} \ <\text{coproc}>\, \{#\}<\text{opc1}>\, \,<\text{Rt}>\, \,<\text{CRn}>\, \,<\text{CRm}>\, \{\#\}<\text{opc2}>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return HAMAIR1;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    return HAMAIR1;
end;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  HAMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HAMAIR1 = R[t];

The HCPTR characteristics are:

**Purpose**

**Controls:**

- Trapping to Hyp mode of Non-secure access, at EL1 or EL0, to trace, and to Advanced SIMD and floating-point functionality.
- Hyp mode access to trace, and to Advanced SIMD and floating-point functionality.

**Note**

Accesses to this functionality:

- From Non-secure modes other than Hyp mode are also affected by settings in the CPACR and NSACR.
- From Hyp mode are also affected by settings in the NSACR.

Exceptions generated by the CPACR and NSACR controls are higher priority than those generated by the HCPTR controls.

**Configuration**

AArch32 System register HCPTR bits [31:0] are architecturally mapped to AArch64 System register CPTR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HCPT is a 32-bit register.

**Field descriptions**

The HCPTR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| TCPAC | TAM | RES0 | TTA | RES0 | TASE | RES0 | RES1 | TCP11 | TCP10 | RES1 |

**TCPAC, bit [31]**

Traps Non-secure EL1 accesses to the CPACR to Hyp mode.

<table>
<thead>
<tr>
<th>TCPAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to the CPACR are trapped to Hyp mode.</td>
</tr>
</tbody>
</table>

**Note**

The CPACR is not accessible at EL0.

In a system where the PE resets into EL2 or EL3, this field resets to 0.
**TAM, bit [30]**

When AMUv1 is implemented:

Trap Activity Monitor access. Traps Non-secure EL1 and EL0 accesses to all Activity Monitor registers to EL2.

<table>
<thead>
<tr>
<th>TAM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are not trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are trapped to Hyp mode.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bits [29:21]**

Reserved, RES0.

**TTA, bit [20]**

Traps Non-secure System register accesses to all implemented trace registers to Hyp mode.

<table>
<thead>
<tr>
<th>TTA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>
| 0b1  | Any Non-secure System register access to an implemented trace register is trapped to Hyp mode, unless the access is trapped to EL1 by a CPACR or NSACR control, or the access is from Non-secure EL0 and the definition of the register in the appropriate trace architecture specification indicates that the register is not accessible from EL0. A trapped instruction generates:  
  • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1.  
  • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode. |

If the implementation does not include a PE trace unit, or does not include a System register interface to the PE trace unit registers, it is IMPLEMENTATION DEFINED whether this bit:

- Is RES0.
- Is RES1.
- Can be written from Hyp mode, and from Secure Monitor mode when SCR.NS is 1.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSTRCDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

**Note**

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the implementation includes an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED, and a resulting Undefined Instruction exception is higher priority than a HCPTR.TTA Hyp Trap exception.
- The architecture does not provide traps on trace register accesses through the optional memory-mapped debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

In a system where the PE resets into EL2 or EL3, this field resets to 0.
Bits [19:16]
Reserved, RES0.

TASE, bit [15]
Traps Non-secure execution of Advanced SIMD instructions to Hyp mode when the value of HC PTR.TCP10 is 0.

<table>
<thead>
<tr>
<th>TASE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>When the value of HC PTR.TCP10 is 0, any attempt to execute an Advanced SIMD instruction in Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates:</td>
</tr>
<tr>
<td></td>
<td>• A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1.</td>
</tr>
<tr>
<td></td>
<td>• An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.</td>
</tr>
</tbody>
</table>

When the value of HC PTR.TCP10 is 1, the value of this field is ignored.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, then it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of NSACR.NSASEDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see ‘Controls of Advanced SIMD operation that do not apply to floating-point operation’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

Bit [14]
Reserved, RES0.

Bits [13:12]
Reserved, RES1.

TCP11, bit [11]
The value of this field is ignored. If this field is programmed with a different value to the TCP10 bit then this field is UNKNOWN on a direct read of the HC PTR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of NSACR.cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

TCP10, bit [10]
Trap Non-secure accesses to Advanced SIMD and floating-point functionality to Hyp mode:
Meaning

<table>
<thead>
<tr>
<th>TCP10</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempted access to Advanced SIMD and floating-point functionality from Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates:</td>
</tr>
<tr>
<td></td>
<td>• A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1.</td>
</tr>
<tr>
<td></td>
<td>• An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode.</td>
</tr>
</tbody>
</table>

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPXCR System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of NSACR.cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

Bits [9:0]

Reserved, RES1.

Accessing the HCPTTR

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
else
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return HCPTTR;
elsif PSTATE.EL == EL2 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HCPTTR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HCPTTR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3_TCPAC == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    HCPTR = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HCPTR = R[t];
HCR, Hyp Configuration Register

The HCR characteristics are:

**Purpose**

Provides configuration controls for virtualization, including defining whether various Non-secure operations are trapped to Hyp mode.

**Configuration**

AArch32 System register HCR bits [31:0] are architecturally mapped to AArch64 System register \texttt{HCR\_EL2[31:0]}.

If EL2 is not implemented, this register is \texttt{RES0} from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally \texttt{UNKNOWN} values.

**Attributes**

HCR is a 32-bit register.

**Field descriptions**

The HCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>TRVM, bit [30]</td>
</tr>
<tr>
<td>29</td>
<td>HCD, bit [29]</td>
</tr>
<tr>
<td>28</td>
<td>TGE</td>
</tr>
<tr>
<td>27</td>
<td>TVM</td>
</tr>
<tr>
<td>26</td>
<td>TPU</td>
</tr>
<tr>
<td>25</td>
<td>TPC</td>
</tr>
<tr>
<td>24</td>
<td>TSW</td>
</tr>
<tr>
<td>23</td>
<td>TAC</td>
</tr>
<tr>
<td>22</td>
<td>TIDCP</td>
</tr>
<tr>
<td>21</td>
<td>TSC</td>
</tr>
<tr>
<td>20</td>
<td>TID3</td>
</tr>
<tr>
<td>19</td>
<td>TID2</td>
</tr>
<tr>
<td>18</td>
<td>TID1</td>
</tr>
<tr>
<td>17</td>
<td>TID0</td>
</tr>
<tr>
<td>16</td>
<td>TWE</td>
</tr>
<tr>
<td>15</td>
<td>TWI</td>
</tr>
<tr>
<td>14</td>
<td>DC</td>
</tr>
<tr>
<td>13</td>
<td>BSU</td>
</tr>
<tr>
<td>12</td>
<td>FB</td>
</tr>
<tr>
<td>11</td>
<td>VA</td>
</tr>
<tr>
<td>10</td>
<td>VI</td>
</tr>
<tr>
<td>9</td>
<td>VF</td>
</tr>
<tr>
<td>8</td>
<td>AMO</td>
</tr>
<tr>
<td>7</td>
<td>IMO</td>
</tr>
<tr>
<td>6</td>
<td>FMO</td>
</tr>
<tr>
<td>5</td>
<td>PTW</td>
</tr>
<tr>
<td>4</td>
<td>SWIO</td>
</tr>
<tr>
<td>3</td>
<td>VM</td>
</tr>
<tr>
<td>2</td>
<td>CONTEXTIDR</td>
</tr>
</tbody>
</table>

**TRVM, bit [30]**

Trap Reads of Virtual Memory controls. Traps Non-secure EL1 reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state.

The registers for which read accesses are trapped are as follows:

- SCTLR
- TTBR0, TTBR1
- TTBCR
- TTBCR2
- DACR
- DFSR
- IFSR
- DFAR
- IFAR
- ADFSR
- AIFSR
- PRRR
- NMRR
- MAIR0
- MAIR1
- AMAIR0
- AMAIR1
- CONTEXTIDR

<table>
<thead>
<tr>
<th>TRVM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 read accesses to the specified Virtual Memory controls are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**HCD, bit [29]**

HVC instruction disable. Disables Non-secure EL1 and EL2 execution of HVC instructions, when EL2 is enabled in the current Security state.
### HCD

<table>
<thead>
<tr>
<th>HCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>HVC instruction execution is enabled at EL2 and EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>HVC instructions are UNDEFINED at EL2 and Non-secure EL1. The Undefined Instruction exception is taken to the Exception level at which the HVC instruction is executed.</td>
</tr>
</tbody>
</table>

**Note**

HVC instructions are always UNDEFINED at EL0.

This bit is only implemented if EL3 is not implemented. Otherwise, it is RES0.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

### Bit [28]

Reserved, RES0.

### TGE, bit [27]

Trap General Exceptions, from Non-secure EL0.

<table>
<thead>
<tr>
<th>TGE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on execution at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, then:</td>
</tr>
<tr>
<td></td>
<td>• All exceptions that would be routed to EL1 are routed to EL2.</td>
</tr>
<tr>
<td></td>
<td>• The SCTLR.M bit is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR.</td>
</tr>
<tr>
<td></td>
<td>• The HCR.{FMO,IMO,AMO} bits are treated as being 1 for all purposes other than returning the result of a direct read of HCR.</td>
</tr>
<tr>
<td></td>
<td>• All virtual interrupts are disabled.</td>
</tr>
<tr>
<td></td>
<td>• Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled.</td>
</tr>
<tr>
<td></td>
<td>• An exception return to EL1 is treated as an illegal exception return.</td>
</tr>
<tr>
<td></td>
<td>• Monitor mode execution of an MSR or CPS instruction that changes CPSR.M to a Non-secure EL1 mode is an illegal change to PSTATE.M. For more information see 'Illegal changes to PSTATE.M' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers' Model).</td>
</tr>
</tbody>
</table>

Also, when HCR.TGE is 1:

- If EL3 is using AArch32, an attempt to change from a Secure PL1 mode to a Non-secure EL1 mode by changing SCR.NS from 0 to 1 results in SCR.NS remaining as 0.
- The HDCR.{TDRA, TDOSA, TDA, TDE} bits are ignored and treated as being 1 other than for the purpose of a direct read of HDCR.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

### TVM, bit [26]

Trap Virtual Memory controls. Traps Non-secure EL1 writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state.

The registers for which write accesses are trapped are as follows:

SCTLR, TTBR0, TTBR1, TTBCR, TTBCR2, DACR, DFSR, IPSR, DFAR, IFAR, ADFSR, AIFSRR, PRRR, NMRR, MAIR0, MAIR1, AMAIR0, AMAIR1, CONTEXTIDR.
TVM, bit [25]

This control does not cause any instructions to be trapped.

Non-secure EL1 write accesses to the specified virtual memory control registers are trapped to EL2.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

TTLB, bit [25]

Trap TLB maintenance instructions. Traps Non-secure EL1 execution of a TLBI instruction to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- TLBIAVALL, TLBIMVAIS, TLBIAVAIN, TLBVAIS, TLBVAIM, ITLBIALL, ITLBMVA, ITLBIASID, ITLBIALL, ITLBMVAA, ITLBIMVAA, ITLBIMVAAL, TLBIMVAAL

TTLB, bit [25]

This control does not cause any instructions to be trapped.

Non-secure EL1 accesses to the specified TLB maintenance instructions are trapped to EL2.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps Non-secure EL1 execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- ICIMVAL, ICIAVALL, ICIAVALLUIS, DCIMVAU.

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

TPU, bit [24]

This control does not cause any instructions to be trapped.

Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

TPC, bit [23]

Trap data or unified cache maintenance instructions that operate to the Point of Coherency. Traps Non-secure EL1 execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- DCIMVAU, DCCIMVAU, DCCIMVAU.

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.
If the Point of Coherency is before any level of data cache, it is implementation defined whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TSW, bit [22]**

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps Non-secure EL1 execution of those cache maintenance instructions by set/way to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- DCISW, DCCSW, DCCISW.

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always undefined at EL0.

<table>
<thead>
<tr>
<th>TSW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TAC, bit [21]**

Trap Auxiliary Control Registers. Traps Non-secure EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, from both Execution states.

This applies to the following register accesses:

ACTLR and, if implemented, ACTLR2.

<table>
<thead>
<tr>
<th>TAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to the specified registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TIDCP, bit [20]**

Trap implementation defined functionality. Traps Non-secure EL1 accesses to the encodings for implementation defined System Registers to EL2, when EL2 is enabled in the current Security state.

MCR and MRC instructions accessing the following encodings:

- All coproc==p15, CRn==c9, Opcode1 = {0-7}, CRm == {c0-c2, c5-c8}, opcode2 == {0-7}.
- All coproc==p15, CRn==c10, Opcode1 =={0-7}, CRm == {c0, c1, c4, c8}, opcode2 == {0-7}.
- All coproc==p15, CRn==c11, Opcode1=={0-7}, CRm == {c0-c8, c15}, opcode2 == {0-7}.

When HCR.TIDCP is set to 1, it is implementation defined whether any of this functionality accessed from Non-secure EL0 is trapped to EL2. Otherwise, it is undefined and the PE takes an Undefined Instruction exception to Non-secure Undefined mode.
### TIDCP

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to the specified System register encodings for IMPLEMENTATION DEFINED functionality are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

### TSC, bit [19]

Traps SMC instructions. Traps Non-secure EL1 execution of SMC instructions to Hyp mode.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute an SMC instruction at Non-secure EL1 is trapped to Hyp mode, regardless of the value of SCR.SCD.</td>
</tr>
</tbody>
</table>

The Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions.

**Note**
- This trap is only implemented if the implementation includes EL3.
- SMC instructions are always UNDEFINED at PL0.
- This bit traps execution of the SMC instruction. It is not a routing control for the SMC exception. Hyp Trap exceptions and SMC exceptions have different preferred return addresses.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

### TID3, bit [18]

Traps Non-secure EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state, as follows:

- ID_PFR0, ID_PFR1, ID_DFR0, ID_AFR0, ID_MMFR0, ID_MMFR1, ID_MMFR2, ID_MMFR3, ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, ID_ISAR5, MVFR0, MVFR1, MVFR2.
- ID_MMFR4 and ID_MMFR5 are trapped unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_MMFR4 or ID_MMFR5 are trapped.
- ID_ISAR6 is trapped unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_ISAR6 are trapped.
- ID_DFR1 is trapped unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to ID_DFR1 are trapped.
- It is IMPLEMENTATION DEFINED whether this bit traps MRC accesses to registers not already mentioned, with coproc==p15, opc1 == 0, CRn == c0, CRm == \{c2-c7\}, opc2 == \{0-7\}.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified Non-secure EL1 read accesses to ID group 3 registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

### TID2, bit [17]

Trap ID group 2. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

- Non-secure EL1 and EL0 reads of the CTR, CCSIDR, CCSIDR2, CLIDR, and CSSEL.R.
- Non-secure EL1 and EL0 writes to the CSSEL.R.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified Non-secure EL1 and EL0 accesses to ID group 2 registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.
**TID1, bit [16]**

Trap ID group 1. Traps Non-secure EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state:

- **TCMTR**, **TLBTR**, **REVIDR**, **AIDR**.

<table>
<thead>
<tr>
<th>TID1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified Non-secure EL1 read accesses to ID group 1 registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TID0, bit [15]**

Trap ID group 0. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

- Non-secure EL1 reads of the **JIDR** and **FPSID**.
- If the **JIDR** is RAZ from Non-secure EL0, Non-secure EL0 reads of the **JIDR**.

**Note**

- It is IMPLEMENTATION DEFINED whether the **JIDR** is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0 then the Undefined Instruction exception takes precedence over this trap.
- The **FPSID** is not accessible at EL0.
- Writes to the **FPSID** are ignored, and not trapped by this control.

<table>
<thead>
<tr>
<th>TID0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified Non-secure EL1 read accesses to ID group 0 registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TWE, bit [14]**

Traps Non-secure EL0 and EL1 execution of WFE instructions to EL2, when EL2 is enabled in the current Security state.

<table>
<thead>
<tr>
<th>TWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFE instruction at Non-secure EL0 or EL1 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE.</td>
</tr>
</tbody>
</table>

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TWI, bit [13]**

Traps Non-secure EL0 and EL1 execution of WFI instructions to EL2, when EL2 is enabled in the current Security state.
TWI

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**DC, bit [12]**

Default Cacheability.

<table>
<thead>
<tr>
<th>DC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the Non-secure EL1&amp;0 translation regime.</td>
</tr>
<tr>
<td>0b1</td>
<td>In Non-secure state:</td>
</tr>
<tr>
<td></td>
<td>• The SCTLR.M field behaves as 0 for all purposes other than a direct read of the value of the field.</td>
</tr>
<tr>
<td></td>
<td>• The HCR.VM field behaves as 1 for all purposes other than a direct read of the value of the field.</td>
</tr>
<tr>
<td></td>
<td>• The memory type produced by the first stage of the EL1&amp;0 translation regime is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-Allocate, Outer Write-Back Read-Allocate Write-Allocate.</td>
</tr>
</tbody>
</table>

This field has no effect on the EL2 and EL3 translation regimes.

This field is permitted to be cached in a TLB.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**BSU, bits [11:10]**

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from Non-secure EL1 or Non-secure EL0:

<table>
<thead>
<tr>
<th>BSU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Full system.</td>
</tr>
</tbody>
</table>

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**FB, bit [9]**

Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from Non-secure EL1:

- BPIALL, TLBIALL, TLBIMVA, TLBIASID, DTLBIALL, DTLBIMVA, DTLBIASID, ITLBIALL, ITLBMVA, ITLBIASID, TLBIMVAA, ICIALLU, TLBIMVAL, TLBIMVAAL.
Meaning

0b0 This field has no effect on the operation of the specified instructions.

0b1 When one of the specified instruction is executed at Non-secure EL1, the instruction is broadcast within the Inner Shareable shareability domain.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**VA, bit [8]**

Virtual SError interrupt exception.

<table>
<thead>
<tr>
<th>VA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This mechanism is not making a virtual SError interrupt pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>A virtual SError interrupt is pending because of this mechanism.</td>
</tr>
</tbody>
</table>

The virtual SError interrupt is enabled only when the value of HCR.{TGE, AMO} is \{0, 1\}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**VI, bit [7]**

Virtual IRQ exception.

<table>
<thead>
<tr>
<th>VI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This mechanism is not making a virtual IRQ pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>A virtual IRQ is pending because of this mechanism.</td>
</tr>
</tbody>
</table>

The virtual IRQ is enabled only when the value of HCR.{TGE, IMO} is \{0, 1\}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**VF, bit [6]**

Virtual FIQ exception.

<table>
<thead>
<tr>
<th>VF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This mechanism is not making a virtual FIQ pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>A virtual FIQ is pending because of this mechanism.</td>
</tr>
</tbody>
</table>

The virtual FIQ is enabled only when the value of HCR.{TGE, FMO} is \{0, 1\}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**AMO, bit [5]**

SError interrupt Mask Override. When this bit is set to 1, it overrides the effect of CPSR.A, and enables virtual exception signaling by the VA bit.

If the value of HCR.TGE is 0, then virtual SError interrupts are enabled in Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.AMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

In a system where the PE resets into EL2 or EL3, this field resets to 0.
IMO, bit [4]

IRQ Mask Override. When this bit is set to 1, it overrides the effect of CPSR.I, and enables virtual exception signaling by the VI bit.

If the value of HCR.TGE is 0, then Virtual IRQ interrupts are enabled in the Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.IMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

FMO, bit [3]

FIQ Mask Override. When this bit is set to 1, it overrides the effect of CPSR.F, and enables virtual exception signaling by the VF bit.

If the value of HCR.TGE is 0, then Virtual FIQ interrupts are enabled in the Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.FMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

PTW, bit [2]

Protected Table Walk. In the Non-secure PL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the two stages of translation means the access might be made to a type of Device memory. If this occurs then the value of this bit determines the behavior:

<table>
<thead>
<tr>
<th>PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively.</td>
</tr>
<tr>
<td>0b1</td>
<td>The memory access generates a stage 2 Permission fault.</td>
</tr>
</tbody>
</table>

This field is permitted to be cached in a TLB.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

SWIO, bit [1]

Set/Way Invalidation Override. Causes Non-secure EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way.

<table>
<thead>
<tr>
<th>SWIO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the operation of data cache invalidate by set/way instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way.</td>
</tr>
</tbody>
</table>

When this bit is set to 1, DCISW performs the same invalidation as a DCCISW instruction.

As a result of changes to the behavior of DCISW, this bit is redundant in Armv8. This bit can be implemented as RES1.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the Non-secure EL1&0 translation regime.

<table>
<thead>
<tr>
<th>VM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1&amp;0 stage 2 address translation disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1&amp;0 stage 2 address translation enabled.</td>
</tr>
</tbody>
</table>
If the HCR.DC bit is set to 1, then the behavior of the PE when executing in a Non-secure mode other than Hyp mode is consistent with HCR.VM being 1, regardless of the actual value of HCR.VM, other than the value returned by an explicit read of HCR.VM.

When the value of this bit is 1, data cache invalidate instructions executed at Non-secure EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR.SWIO bit.

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**Accessing the HCR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
else
  UNDEFINED;
elifs PSTATE.EL == EL2 then
  return HCR;
elifs PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
elelse
    return HCR;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
else
  UNDEFINED;
elifs PSTATE.EL == EL2 then
  HCR = R[t];
elifs PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
elelse
    HCR = R[t];
```
HCR2, Hyp Configuration Register 2

The HCR2 characteristics are:

**Purpose**

Provides additional configuration controls for virtualization.

**Configuration**

AArch32 System register HCR2 bits [31:0] are architecturally mapped to AArch64 System register HCR_EL2[63:32].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HCR2 is a 32-bit register.

**Field descriptions**

The HCR2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>TTLBIS</td>
<td>RES0</td>
<td>TOCU</td>
<td>RES0</td>
<td>TICAB</td>
<td>TID4</td>
<td>RES0</td>
<td>MIOCNCE</td>
<td>TEA</td>
<td>TERR</td>
<td>RES0</td>
<td>ID</td>
<td>CD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:23]**

Reserved, RES0.

**TTLBIS, bit [22]**

When ARMv8.2-EVT is implemented:

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of the following TLB maintenance instructions at EL1 to EL2:

- TLBIALLIS
- TLBIMVAIS
- TLBIASIDIS
- TLBIMVAIS
- TLBIMVALIS
- TLBIMVALIS

<table>
<thead>
<tr>
<th>TTLBIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 execution of the specified TLB maintenance instructions is trapped to EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-EVT is not implemented, this field is RES0.

When ARMv8.1-VHE and the value of HCR_EL2.E2H, TGE is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

**Otherwise:**

Reserved, RES0.
Bit [21]

Reserved, RES0.

TOCU, bit [20]

When ARMv8.2-EVT is implemented:

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2.

This applies to the following instructions:

- When Non-secure EL0 is using AArch64, **IC IVAU**, **DC CVAU**. However, if the value of **SCTLR_EL1**.UCI is 0 these instructions are **UNDEFINED** at EL0 and any resulting exception is higher priority than this trap to EL2.
- When EL1 is using AArch64, **IC IVAU**, **IC IALLU**, **DC CVAU**.
- When Non-secure EL1 is using AArch32, **ICIMVAU**, **ICIALLU**, **DCCMVAU**.

Note

An exception generated because an instruction is **UNDEFINED** at EL0 is higher priority than this trap to EL2. In addition:

- **IC IALLUIS** and **IC IALLU** are always **UNDEFINED** at EL0 using AArch64.
- **ICIMVAU**, **ICIALLU**, **ICIALLUI**, and **DCCMVAU** are always **UNDEFINED** at EL0 using AArch32.

<table>
<thead>
<tr>
<th><strong>TOCU</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure execution of the specified cache maintenance instructions is trapped to EL2.</td>
</tr>
</tbody>
</table>

If ARMv8.2-EVT is not implemented, this field is RES0.

If the Point of Unification is before any level of data cache, it is **IMPLEMENTATION DEFINED** whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is **IMPLEMENTATION DEFINED** whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When ARMv8.1-VHE is implemented, and the value of **HCR_EL2**.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

TICAB, bit [18]

When ARMv8.2-EVT is implemented:

Trap ICIALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 to EL2.

This applies to the following instructions:

**ICIALLUI**.
TICAB  

<table>
<thead>
<tr>
<th>TICAB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2.</td>
</tr>
</tbody>
</table>

If Armv8.2-EVT is not implemented, this field is RES0.

If the Point of Unification is before any level of instruction cache, it is implementation defined whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When Armv8.1-VHE and the value of HCR_EL2.(E2H, TGE) is \{1, 1\}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

TID4, bit [17]

When Armv8.2-EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2:

- EL1 reads of CCSIDR, CCSIDR2, CLIDR, and CSSELR.
- EL1 writes to CSSELR.

<table>
<thead>
<tr>
<th>TID4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>The specified Non-secure EL1 and EL0 accesses to ID group 4 registers are trapped to EL2.</td>
</tr>
</tbody>
</table>

If Armv8.2-EVT is not implemented, this field is RES0.

When Armv8.1-VHE is implemented and the value of HCR_EL2.(E2H, TGE) is \{1, 1\}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

Bits [16:7]

Reserved, RES0.

MIOCNCE, bit [6]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the Non-secure PL1&0 translation regime.

<table>
<thead>
<tr>
<th>MIOCNCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For the Non-secure PL1&amp;0 translation regime, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there must be no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.</td>
</tr>
<tr>
<td>0b1</td>
<td>For the Non-secure PL1&amp;0 translation regime, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there might be a loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute.</td>
</tr>
</tbody>
</table>

For more information see 'Mismatched memory attributes' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E2 (The AArch32 Application Level Memory Model).
This field can be implemented as RAZ/WI.

In a system where the PE resets into EL2 or EL3, this field resets to an architecturally **UNKNOWN** value.

**TEA, bit [5]**

Route synchronous External abort exceptions from EL0 and EL1 to EL2. If the RAS Extension is implemented, the possible values of this bit are:

<table>
<thead>
<tr>
<th>TEA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not route synchronous External abort exceptions from Non-secure EL0 and EL1 to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Route synchronous External abort exceptions from Non-secure EL0 and EL1 to EL2, if not routed to EL3.</td>
</tr>
</tbody>
</table>

When the RAS Extension is not implemented, this field is **RES0**.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**TERR, bit [4]**

**When RAS is implemented:**

Trap Error record accesses from EL1 to EL2. Trap accesses to the following registers from EL1 to EL2:

- ERRIDR
- ERRSELR
- ERXADDR
- ERXADDR2
- ERXCTRL
- ERXCTRL2
- ERXFR
- ERXFR2
- ERXCTLR
- ERXCTLR2
- ERXFR
- ERXFR2
- ERXCTLR
- ERXCTLR2
- ERXSTATUS
- ERXMISR4
- ERXMISR5
- ERXMISR6
- ERXMISR7

<table>
<thead>
<tr>
<th>TERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to the specified registers from EL1 generate a Trap exception to EL2.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, **RES0**.

**Bits [3:2]**

Reserved, **RES0**.

**ID, bit [1]**

Stage 2 Instruction access cacheability disable. For the Non-secure PL1&0 translation regime, when **HCR.VM==1**, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

<table>
<thead>
<tr>
<th>ID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on stage 2 of the Non-secure PL1&amp;0 translation regime.</td>
</tr>
<tr>
<td>0b1</td>
<td>For the Non-secure PL1&amp;0 translation regime, forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.</td>
</tr>
</tbody>
</table>

This bit has no effect on the EL2 translation regime.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**CD, bit [0]**

Stage 2 Data access cacheability disable. When **HCR.VM==1**, this forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable for the Non-secure PL1&0 translation regime.
This control has no effect on stage 2 of the Non-secure PL1&0 translation regime for data accesses and translation table walks.

For the Non-secure PL1&0 translation regime, forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

This bit has no effect on the EL2 translation regime.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**Accessing the HCR2**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return HCR2;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HCR2 = R[t];
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  HCR2 = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HCR2 = R[t];
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
The HDCR characteristics are:

**Purpose**

Controls the trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to functions provided by the debug and trace architectures and the Performance Monitors Extension.

**Configuration**

AArch32 System register HDCR bits [31:0] are architecturally mapped to AArch64 System register MDCR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3, and other than for a direct read of the register, the PE behaves as if HDCR.HPMN == PMCR.N.

This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HDCR is a 32-bit register.

**Field descriptions**

The HDCR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | MTPME | TDCCH | HPRES0 | HCCD | RES0 | TTRFRES0 | HPMD | RES0 | TDRATDOSA | DATDE | HPME | TPM | TPMCR | HPMD |

**Bits [31:29]**

Reserved, RES0.

**MTPME, bit [28]**

When ARMv8.6-MTPMU is implemented and !HaveEL(EL3):

Multi-threaded PMU Enable. Enables use of the PMEVTYPE<\n> MT bits.

<table>
<thead>
<tr>
<th>MTPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ARMv8.6-MTPMU is disabled. The Effective value of PMEVTYPE&lt;\n&gt; MT is zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>PMEVTYPE&lt;\n&gt; MT bits not affected by this bit.</td>
</tr>
</tbody>
</table>

If ARMv8.6-MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this bit is 0.

On a Cold reset, in a system where the PE resets into EL2 or EL3, this field resets to 1.

**Otherwise:**

Reserved, RES0.
**TDCC, bit [27]**

When ARMv8.6-FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

<table>
<thead>
<tr>
<th>TDCC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any register accesses to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Hyp Trap exception, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.</td>
</tr>
</tbody>
</table>

The DCC registers trapped by this control are:

- DBGDTRRXext, DBGDTRTXext, DBGDSCRint, DBGDCINT, and, when the PE is in Non-debug state, DBGDTRRXint and DBGDTRTXint.

The traps are reported with EC syndrome value:

- 0x05 for trapped MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped LDC to DBGDTRTXint and STC from DBGDTRRXint.

When the PE is in Debug state, HDCR.TDCC does not trap any accesses to:

- DBGDTRRXint and DBGDTRTXint.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**HLP, bit [26]**

When ARMv8.5-PMU is implemented:

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

<table>
<thead>
<tr>
<th>HLP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR&lt;n&gt;[31:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR&lt;n&gt;[63:0].</td>
</tr>
</tbody>
</table>

If the highest implemented Exception level is using AArch32, it is **IMPLEMENTATION DEFINED** whether this bit is read/write or RAZ/WI.

If HDCR.HPMN is less than PMCR.N, this bit affects the operation of event counters in the range [HDCR.HPMN..(PMCR.N-1)]. Otherwise this bit has no effect on the operation of the event counters.

**Note**

The effect of HDCR.HPMN on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, and regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the HDCR.HPMN field.

**Note**
PMECNTR cannot be accessed directly in AArch32 state.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bits [25:24]**

Reserved, RES0.

**HCCD, bit [23]**

When ARMv8.5-PMU is implemented:

Hypervisor Cycle Counter Disable. Prohibits PMCCNTR from counting at EL2.

<table>
<thead>
<tr>
<th>HCCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting by PMCCNTR is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting by PMCCNTR is prohibited at EL2.</td>
</tr>
</tbody>
</table>

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bits [22:20]**

Reserved, RES0.

**TTRF, bit [19]**

When ARMv8.4-Trace is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2.

<table>
<thead>
<tr>
<th>TTRF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to TRFCR at EL1 are not affected by this control bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to TRFCR at EL1 generate a Hyp Trap exception.</td>
</tr>
</tbody>
</table>

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bit [18]**

Reserved, RES0.
**HPMD, bit [17]**

*When ARMv8.1-PMU is implemented:*

Guest Performance Monitors Disable. This control prohibits event counting at EL2.

<table>
<thead>
<tr>
<th>HPMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counting allowed in Hyp mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counting prohibited in Hyp mode.</td>
</tr>
<tr>
<td></td>
<td>If ARMv8.2-Debug is not implemented, event counting is</td>
</tr>
<tr>
<td></td>
<td>prohibited unless enabled by the IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td></td>
<td>authentication interface</td>
</tr>
<tr>
<td></td>
<td>ExternalSecureNoninvasiveDebugEnabled()</td>
</tr>
</tbody>
</table>

This control applies only to:

- The event counters in the range [0..(HDCR.HPMN-1)].
- If PMCR.DP is set to 1, PMCCNTR.

The other event counters are unaffected. When PMCR.DP is set to 0, PMCCNTR is unaffected.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bits [16:12]**

Reserved, RES0.

**TDRA, bit [11]**

Trap Debug ROM Address register access. Traps Non-secure EL0 and EL1 System register accesses to the Debug ROM registers to Hyp mode.

<table>
<thead>
<tr>
<th>TDRA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL0 and EL1 System register accesses to the DBGDRAR or DBGDSAR are trapped to Hyp mode, unless it is trapped by DBGDSCRext.UDCCdis.</td>
</tr>
</tbody>
</table>

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**TDOSA, bit [10]**

*When ARMv8.0-DoubleLock is implemented:*

Trap debug OS-related register access. Traps Non-secure EL1 System register accesses to the powerdown debug registers to Hyp mode.

<table>
<thead>
<tr>
<th>TDOSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode.</td>
</tr>
</tbody>
</table>

The registers for which accesses are trapped are as follows:

- DBGOSLAR, DBGOSLAR, DBGOSDLR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

**Note**
These registers are not accessible at EL0.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Trap debug OS-related register access. Traps Non-secure EL1 System register accesses to the powerdown debug registers to Hyp mode.

<table>
<thead>
<tr>
<th>TDOSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode.</td>
</tr>
</tbody>
</table>

The registers for which accesses are trapped are as follows:

- DBGOSLSR, DBGOSLR, and DBGPRCR.
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to DBGOSDLR are trapped.

Note

These registers are not accessible at EL0.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDA, bit [9]

Trap debug access. Traps Non-secure EL0 and EL1 System register accesses to those debug System registers in the (coproc==0b1110) encoding space that are not trapped by either of the following:

- HDCR.TDRA.
- HDCR.TDOSA.

<table>
<thead>
<tr>
<th>TDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL0 or EL1 System register accesses to the debug registers, other than the registers trapped by HDCR.TDRA and HDCR.TDOSA, are trapped to Hyp mode, unless it is trapped by DBGDSCRxext.UDCCdis.</td>
</tr>
</tbody>
</table>

Traps of AArch32 accesses to DBGDTRRXint and DBGDTRTXint are ignored in Debug state.

If HCR.TGE or HDCR.TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDE, bit [8]

Trap Debug exceptions. The possible values of this bit are:

<table>
<thead>
<tr>
<th>TDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on the routing of debug exceptions, and has no effect on Non-secure accesses to debug registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>Debug exceptions generated at EL1 or EL0 are routed to EL2 when enabled in the current Security state. The HDCR.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register.</td>
</tr>
</tbody>
</table>
When \( \text{HCR.TGE} = 1 \), the PE behaves as if the value of this field is 1 for all purposes other than returning the value of a direct read of the register.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**HPME, bit [7]**

*When PMUv3 is implemented:*

[\( \text{HDCR.HPMN..(N-1)} \)] event counters enable.

<table>
<thead>
<tr>
<th>HPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counters in the range [( \text{HDCR.HPMN..(PMCR.N-1)} )] are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counters in the range [( \text{HDCR.HPMN..(PMCR.N-1)} )] are enabled by ( \text{PMCNTENSET} ).</td>
</tr>
</tbody>
</table>

If \( \text{HDCR.HPMN} \) is less than \( \text{PMCR.N} \), the event counters in the range [\( \text{HDCR.HPMN..(PMCR.N-1)} \)], are enabled and disabled by this bit. Otherwise this bit has no effect on the operation of the event counters.

**Note**

The effect of \( \text{HDCR.HPMN} \) on the operation of this bit applies regardless of whether EL2 is enabled in the current Security state.

For more information see the description of the HPMN field.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally \text{UNKNOWN} value.

**Otherwise:**

Reserved, \text{RES0}.

**TPM, bit [6]**

*When PMUv3 is implemented:*

Trap Performance Monitors accesses. Traps Non-secure EL0 and EL1 accesses to all Performance Monitors registers to Hyp mode.

<table>
<thead>
<tr>
<th>TPM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL0 and EL1 accesses to all Performance Monitors registers are trapped to Hyp mode.</td>
</tr>
</tbody>
</table>

**Note**

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, \text{RES0}.

**TPMCR, bit [5]**

*When PMUv3 is implemented:*

Trap \( \text{PMCR} \) accesses. Traps Non-secure EL0 and EL1 accesses to the \( \text{PMCR} \) to Hyp mode.
**TPMCR**

<table>
<thead>
<tr>
<th>Meanings</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Note**

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**HPMN, bits [4:0]**

When PMUv3 is implemented:

Defines the number of event counters that are accessible from Non-secure EL1 modes, and from Non-secure EL0 modes if unprivileged access is enabled.

If HPMN is less than **PMCR.N**, HPMN divides the event counters into two ranges, [0..(HPMN-1)] and [HPMN..(**PMCR.N-1**)].

For an event counter in the range [0..(HPMN-1)]:

- The counter is accessible from EL1 and EL2, and from EL0 if unprivileged access to the counters is enabled.
- If ARMv8.5-PMU is implemented, **PMCR.LP** determines whether the counter overflows at **PMEVCNTR<n>[31:0]** or **PMEVCNTR<n>[63:0]**.
- **PMCR.E** enables the operation of counters in this range.

**Note**

If HPMN is equal to **PMCR.N**, this applies to all event counters.

If HPMN is less than **PMCR.N**, for an event counter in the range [HPMN..(**PMCR.N-1**)]:

- The counter is accessible only from EL2 and from Secure state.
- If ARMv8.5-PMU is implemented, **HDCR.HLP** determines whether the counter overflows at **PMEVCNTR<n>[31:0]** or **PMEVCNTR<n>[63:0]**.
- **HDCR.HPME** enables the operation of counters in this range.

If this field is set to 0, or to a value larger than **PMCR.N**, then the following **CONSTRAINED UNPREDICTABLE** behaviors apply:

- The value returned by a direct read of **HDCR.HPMN** is **UNKNOWN**.
- Either:
  - An **UNKNOWN** number of counters are reserved for EL2 use. That is, the PE behaves as if **HDCR.HPMN** is set to an **UNKNOWN** non-zero value less than or equal to **PMCR.N**.
  - All counters are reserved for EL2 use, meaning no counters are accessible from Non-secure EL1 and Non-secure EL0.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in **PMCR.N**.

**Otherwise:**

Reserved, RES0.

**Accessing the HDCR**

Accesses to this register use the following encodings:
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() \&\& ELUsingAArch32(EL2) \&\& HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) \&\& !ELUsingAArch32(EL3) \&\& MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HDCR = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDCR = R[t];
    end
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() \&\& ELUsingAArch32(EL2) \&\& HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) \&\& !ELUsingAArch32(EL3) \&\& MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HDCR = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDCR = R[t];
    end

27/09/2019 18:22; 6134483bd14dc8c12a99c984chb6e3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
HDFAR, Hyp Data Fault Address Register

The HDFAR characteristics are:

**Purpose**

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception that is taken to Hyp mode.

**Configuration**

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch64 System register `FAR_EL2[31:0]`.

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch32 System register `DFAR[31:0] (S)` when `HaveEL(EL2)`, `HaveEL(EL3)` and `HighestELUsingAArch32()`.

If EL2 is not implemented, this register is `RES0` from EL3.

RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**

HDFAR is a 32-bit register.

**Field descriptions**

The HDFAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA of faulting address of synchronous Data Abort exception taken to Hyp mode</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

VA of faulting address of synchronous Data Abort exception taken to Hyp mode.

On a Prefetch Abort exception, this register is `UNKNOWN`.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register `UNKNOWN`.

This field resets to an architecturally `UNKNOWN` value.

**Accessing the HDFAR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() ! ELUsingAArch32(EL2) & HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() & ELUsingAArch32(EL2) & HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HDFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDFAR = R[t];
else
    return HDFAR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() ! ELUsingAArch32(EL2) & HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() & ELUsingAArch32(EL2) & HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HDFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDFAR = R[t];

HIFAR, Hyp Instruction Fault Address Register

The HIFAR characteristics are:

**Purpose**
Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception that is taken to Hyp mode.

**Configuration**
AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch64 System register `FAR_EL2[63:32]`.

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch32 System register `IFAR[31:0] (S)` when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().

If EL2 is not implemented, this register is `RES` from EL3.

RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**
HIFAR is a 32-bit register.

**Field descriptions**
The HIFAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA of faulting address of synchronous Prefetch Abort exception taken to Hyp mode</td>
</tr>
</tbody>
</table>

**Bits [31:0]**
VA of faulting address of synchronous Prefetch Abort exception taken to Hyp mode.

On a Data Abort exception, this register is `UNKNOWN`.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register `UNKNOWN`.

This field resets to an architecturally `UNKNOWN` value.

**Accessing the HIFAR**
Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \ <\text{coproc}, \ \#<\text{opc1}, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}, \ \#, \ #<\text{opc2}>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b0100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    return HIFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HIFAR;
    end
end

MCR{<c>}{<q>} {<coproc>}, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    HIFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HIFAR = R[t];
    end

HMAIR0, Hyp Memory Attribute Indirection Register 0

The HMAIR0 characteristics are:

**Purpose**

Along with [HMAIR1](#), provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations for memory accesses from Hyp mode.

AttrIndx[2] indicates the HMAIR register to be used:

- When AttrIndx[2] is 0, HMAIR0 is used.
- When AttrIndx[2] is 1, [HMAIR1](#) is used.

**Configuration**

AArch32 System register HMAIR0 bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL2[31:0]](#).

If EL2 is not implemented, this register is res0 from EL3.

RW fields in this register reset to architecturally [UNKNOWN](#) values.

**Attributes**

HMAIR0 is a 32-bit register.

**Field descriptions**

The HMAIR0 bit assignments are:

**When TTBCR.EAE == 1:**

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>Attr3</th>
<th>Attr2</th>
<th>Attr1</th>
<th>Attr0</th>
</tr>
</thead>
</table>

**Attr<n>**, bits [8n+7:8n], for n = 0 to 3

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

<table>
<thead>
<tr>
<th>Attr&lt;n&gt;[7:4]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device memory. See encoding of Attr&lt;n&gt;[3:0] for the type of Device memory.</td>
</tr>
<tr>
<td>0b00RW, RW</td>
<td>Normal memory, Outer Write-Through Transient.</td>
</tr>
<tr>
<td>not 0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01RW, RW</td>
<td>Normal memory, Outer Write-Back Transient.</td>
</tr>
<tr>
<td>not 0b01</td>
<td>Normal memory, Outer Write-Through Non-transient.</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Outer Write-Back Non-transient.</td>
</tr>
</tbody>
</table>

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

---

Page 2508
### Attr\(\text{n}\)[3:0]

<table>
<thead>
<tr>
<th>Attr(\text{n})[3:0]</th>
<th>Meaning when Attr(\text{n})[7:4] is 0b0000</th>
<th>Meaning when Attr(\text{n})[7:4] is not 0b0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-(\text{n})GnRnE memory</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td>0b00RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Through Transient</td>
</tr>
<tr>
<td>0b0100</td>
<td>Device-(\text{n})GnRE memory</td>
<td>Normal memory, Inner Non-cacheable</td>
</tr>
<tr>
<td>0b01RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Transient (RW=0b00)</td>
</tr>
<tr>
<td>0b1000</td>
<td>Device-(\text{n})GRE memory</td>
<td>Normal memory, Inner Write-Through Non-transient</td>
</tr>
<tr>
<td>0b10RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>0b1100</td>
<td>Device-GRE memory</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
<tr>
<td>0b11RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
</tbody>
</table>

\(R = \) Inner Read-Allocate policy, \(W = \) Inner Write-Allocate policy.

The \(R\) and \(W\) bits in some Attr\(\text{n}\) fields have the following meanings:

<table>
<thead>
<tr>
<th>R or W</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Allocate</td>
</tr>
<tr>
<td>0b1</td>
<td>Allocate</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

### Accessing the HMAIR0

Accesses to this register use the following encodings:

\[
\text{MRC}\{c\}\{q\} \; <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elseif PSTATE.EL == EL1 then
if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
AArch32.TakeHypTrapException(0x03);
else
UNDEFINED;
elsif PSTATE.EL == EL2 then
return HMAIR0;
elsif PSTATE.EL == EL3 then
if SCR.NS == '0' then
UNDEFINED;
else
return HMAIR0;

\[
\text{MCR}\{c\}\{q\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & & !ELUsingAArch32(EL2) & & HSTR_EL2.T10 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() & & ELUsingAArch32(EL2) & & HSTR.T10 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  HMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HMAIR0 = R[t];
HMAIR1, Hyp Memory Attribute Indirection Register 1

The HMAIR1 characteristics are:

**Purpose**

Along with HMAIR0, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations for memory accesses from Hyp mode.

AttrIndx[2] indicates the HMAIR register to be used:

- When AttrIndx[2] is 0, HMAIR0 is used.
- When AttrIndx[2] is 1, HMAIR1 is used.

**Configuration**

AArch32 System register HMAIR1 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL2[63:32].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HMAIR1 is a 32-bit register.

**Field descriptions**

The HMAIR1 bit assignments are:

**When TTBCR.EAE == 1:**

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Attr7 | Attr6 | Attr5 | Attr4 |

**Attr<n>, bits [8(n-4)+7:8(n-4)], for n = 4 to 7**

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

<table>
<thead>
<tr>
<th>Attr&lt;n&gt;[7:4]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device memory. See encoding of Attr&lt;n&gt;[3:0] for the type of Device memory.</td>
</tr>
<tr>
<td>0b00RW, RW, not 0b00</td>
<td>Normal memory, Outer Write-Through Transient.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01RW, RW, not 0b00</td>
<td>Normal memory, Outer Write-Back Transient.</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Outer Write-Through Non-transient.</td>
</tr>
<tr>
<td>0b11RW</td>
<td>Normal memory, Outer Write-Back Non-transient.</td>
</tr>
</tbody>
</table>

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

...
### Attr<\text{n}>[3:0]  
<table>
<thead>
<tr>
<th>\text{Meaning when Attr&lt;\text{n}&gt;[7:4] is 0b0000}</th>
<th>\text{Meaning when Attr&lt;\text{n}&gt;[7:4] is not 0b0000}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-nGnRnE memory</td>
</tr>
<tr>
<td>0b00RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td>0b0100</td>
<td>Device-nGnRE memory</td>
</tr>
<tr>
<td>0b01RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td>0b1000</td>
<td>Device-nGRE memory</td>
</tr>
<tr>
<td>0b10RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td>0b1100</td>
<td>Device-GRE memory</td>
</tr>
<tr>
<td>0b11RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
</tr>
</tbody>
</table>

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<\text{n}> fields have the following meanings:

<table>
<thead>
<tr>
<th>R or W</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Allocate</td>
</tr>
<tr>
<td>0b1</td>
<td>Allocate</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

### Accessing the HMAIR1

Accesses to this register use the following encodings:

\[ \text{MRC}\{<c>\}{<q>} \ <\text{coproc}, \ {#}\text{opc1}, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}>}, \ {#}\text{opc2}> \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HSTR_EL2.T10 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() \&\& ELUsingAArch32(EL2) \&\& HSTR.T10 == '1' then
      AArch32.TakeHypTrapException(0x03);
   else
      UNDEFINED;
   elsif PSTATE.EL == EL2 then
      return HMAIR1;
   elsif PSTATE.EL == EL3 then
      if SCR.NS == '0' then
         UNDEFINED;
      else
         return HMAIR1;
      end

\[ \text{MCR}\{<c>\}{<q>} \ <\text{coproc}, \ {#}\text{opc1}, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}>}, \ {#}\text{opc2}> \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elself PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end if;
elself PSTATE.EL == EL2 then
    HMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
else
    HMAIR1 = R[t];
end if;
HPFAR, Hyp IPA Fault Address Register

The HPFAR characteristics are:

**Purpose**

Holds the faulting IPA for some aborts on a stage 2 translation taken to Hyp mode.

**Configuration**

AArch32 System register HPFAR bits [31:0] are architecturally mapped to AArch64 System register HPFAR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HPFAR is a 32-bit register.

**Field descriptions**

The HPFAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>FIPA[39:12]</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Execution in any Non-secure mode other than Hyp mode makes this register UNKNOWN.

**FIPA[39:12], bits [31:4]**

Bits [39:12] of the faulting intermediate physical address.

This field resets to an architecturally UNKNOWN value.

**Bits [3:0]**

Reserved, RES0.

**Accessing the HPFAR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HPFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HPFAR;

MCR{<c>{<q}> <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HPFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HPFAR = R[t];
HRMR, Hyp Reset Management Register

The HRMR characteristics are:

**Purpose**

If EL2 is the highest implemented Exception level and this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

**Configuration**

AArch32 System register HRMR bits [31:0] are architecturally mapped to AArch64 System register RMR_EL2[31:0]. Only implemented if EL2 is the highest implemented Exception level. In this case:

- If EL2 can use AArch32 and AArch64 then this register must be implemented.
- If EL2 cannot use AArch64 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HRMR is a 32-bit register.

**Field descriptions**

The HRMR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:2]**

Reserved, RES0.

**RR, bit [1]**

Reset Request. Setting this bit to 1 requests a Warm reset.

This field resets to 0.

**AA64, bit [0]**

When EL2 can use AArch64, determines which Execution state the PE boots into after a Warm reset:

<table>
<thead>
<tr>
<th>AA64</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>AArch64.</td>
</tr>
</tbody>
</table>

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL2 cannot use AArch64 this bit is RAZ/WI.

When implemented as a RW field, this field resets to 0 on a Cold reset.
**Accessing the HRMR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{c\}\{q\} \text{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm}\}, \{#<opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then}
\]
\[
\text{AArch64.AArch32SystemAccessTrap(EL2, 0x03);}
\]
\[
\text{elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAA Arch32(EL2) && HSTR.T12 == '1' then}
\]
\[
\text{AArch32.TakeHypTrapException(0x03);}
\]
\[
\text{elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then}
\]
\[
\text{return HRMR;}
\]
\[
\text{else}
\]
\[
\text{UNDEFINED;}
\]

\[
\text{MCR}\{c\}\{q\} \text{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm}\}, \{#<opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL1 && EL2Enabled() && ELUSingAArch32(EL2) && HSTR_EL2.T12 == '1' then}
\]
\[
\text{AArch64.AArch32SystemAccessTrap(EL2, 0x03);}
\]
\[
\text{elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAA arch32(EL2) && HSTR.T12 == '1' then}
\]
\[
\text{AArch32.TakeHypTrapException(0x03);}
\]
\[
\text{elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then}
\]
\[
\text{HRMR = R[t];}
\]
\[
\text{else}
\]
\[
\text{UNDEFINED;}
\]
HSCTLR, Hyp System Control Register

The HSCTLR characteristics are:

**Purpose**

Provides top level control of the system operation in Hyp mode.

**Configuration**

AArch32 System register HSCTLR bits [31:0] are architecturally mapped to AArch64 System register SCTLR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HSCTLR is a 32-bit register.

**Field descriptions**

The HSCTLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| DSSBS| T32RE| S32RE| ERE| RES0| RES1| WXN| RES0| RES1| RES0| RES0| RES1| RES0| SEDIT| RES0| CP15BEN| RES0| LSMAOE| nTLS|

**DSSBS**, bit [31]

When ARMv8.0-SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

<table>
<thead>
<tr>
<th>DSSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.SSBS is set to 0 on an exception to Hyp mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>PSTATE.SSBS is set to 1 on an exception to Hyp mode.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an IMPLEMENTATION DEFINED value.

**Otherwise:**

Reserved, RES0.

**TE**, bit [30]

T32 Exception Enable. This bit controls whether exceptions to EL2 are taken to A32 or T32 state:

<table>
<thead>
<tr>
<th>TE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exceptions, including reset, taken to A32 state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exceptions, including reset, taken to T32 state.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.
Bits [29:28]
Reserved, RES1.

Bits [27:26]
Reserved, RES0.

EE, bit [25]

The value of the PSTATE.E bit on entry to Hyp mode, the endianness of stage 1 translation table walks in the EL2 translation regime, and the endianness of stage 2 translation table walks in the PL1&0 translation regime.

The possible values of this bit are:

<table>
<thead>
<tr>
<th>EE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Little-endian. PSTATE.E is cleared to 0 on entry to Hyp mode.</td>
</tr>
<tr>
<td></td>
<td>Stage 1 translation table walks in the EL2 translation regime, and</td>
</tr>
<tr>
<td></td>
<td>stage 2 translation table walks in the PL1&amp;0 translation regime are</td>
</tr>
<tr>
<td></td>
<td>little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Big-endian. PSTATE.E is set to 1 on entry to Hyp mode.</td>
</tr>
<tr>
<td></td>
<td>Stage 1 translation table walks in the EL2 translation regime, and</td>
</tr>
<tr>
<td></td>
<td>stage 2 translation table walks in the PL1&amp;0 translation regime are</td>
</tr>
<tr>
<td></td>
<td>big-endian.</td>
</tr>
</tbody>
</table>

If an implementation does not provide Big-endian support at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception Levels higher than EL0, this bit is RES1.

In a system where the PE resets into EL2, this field resets to an implementation defined value.

Bit [24]
Reserved, RES0.

Bits [23:22]
Reserved, RES1.

Bits [21:20]
Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable in the EL2 translation regime is</td>
</tr>
<tr>
<td></td>
<td>forced to XN for accesses from software executing at EL2.</td>
</tr>
</tbody>
</table>

This bit applies only when HSCTRL.M bit is set.

The WXN bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to an architecturally unknown value.

Bit [18]
Reserved, RES1.
Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:13]

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL2:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All instruction access to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. If the value of HSCTLR.M is 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>All instruction access to Normal memory from EL2 can be cached at all levels of instruction and unified cache. If the value of HSCTLR.M is 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.</td>
</tr>
</tbody>
</table>

This bit has no effect on the PL1&0 translation regime.

In a system where the PE resets into EL2, this field resets to 0.

Bit [11]

Reserved, RES1.

Bits [10:9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at EL2.

<table>
<thead>
<tr>
<th>SED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SETEND instruction execution is enabled at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>SETEND instructions are UNDEFINED at EL2.</td>
</tr>
</tbody>
</table>

If the implementation does not support mixed-endian operation at EL2, this bit is RES1.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at EL2.
**ITD**

<table>
<thead>
<tr>
<th>ITD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All IT instruction functionality is enabled at EL2.</td>
</tr>
</tbody>
</table>
| 0b1 | Any attempt at EL2 to execute any of the following is *undefined*:  
  - All encodings of the IT instruction with hw1[3:0]!=1000.  
  - All encodings of the subsequent instruction with the following values for hw1:  
    - 11xxxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM.  
    - 1011xxxxxxxxxxxx: All instructions in Miscellaneous 16-bit instructions.  
    - 10100xxxxxxxxxx: ADD Rd, PC, #imm  
    - 01001xxxxxxxxxx: LDR Rd, [PC, #imm]  
    - 01001xxxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC.  
    - 010001xxxx111111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn.  
  These instructions are always *undefined*, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.  
  It is implementation defined whether the IT instruction is treated as:  
  - A 16-bit instruction, that can only be followed by another 16-bit instruction.  
  - The first half of a 32-bit instruction.  
  This means that, for the situations that are *undefined*, either the second 16-bit instruction or the 32-bit instruction is *undefined*. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction. |

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is constrained unpredictable. For more information see 'Changes to an ITD control by an instruction in an IT block' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.2.4

ITD is optional, but if it is implemented in the *SCTLR* then it must also be implemented in the *HSCTLR*. If it is not implemented then this bit is RAZ/WI.

In a system where the PE resets into EL2, this field resets to an architecturally unknown value.

**Bit [6]**

Reserved, RES0.

**CP15BEN, bit [5]**

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL2:

<table>
<thead>
<tr>
<th>CP15BEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL2 execution of the <em>CP15DMB, CP15DSB</em>, and <em>CP15ISB</em> instructions is <em>undefined</em>.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 execution of the <em>CP15DMB, CP15DSB</em>, and <em>CP15ISB</em> instructions is enabled.</td>
</tr>
</tbody>
</table>

CP15BEN is optional, but if it is implemented in the *SCTLR* then it must also be implemented in the *HSCTLR*. If it is not implemented then this bit is RAO/WI.

In a system where the PE resets into EL2, this field resets to an architecturally unknown value.

**LSMAOE, bit [4]**

*When Armv8.2-LSMAOC is implemented:*

Load Multiple and Store Multiple Atomicity and Ordering Enable.
LSMAOE, bit [0]
For all memory accesses at EL2, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.

<table>
<thead>
<tr>
<th>LSMAOE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For all memory accesses at EL2, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL2 is as defined for Armv8.0.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to 1.

Otherwise:

Reserved, RES1.

nTLSMD, bit [3]
When ARMv8.2-LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

<table>
<thead>
<tr>
<th>nTLSMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL2 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.</td>
</tr>
<tr>
<td>0b1</td>
<td>All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL2 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL2, this field resets to 1.

Otherwise:

Reserved, RES1.

C, bit [2]
Cacheability control, for data accesses at EL2:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All data access to Normal memory from EL2, and all accesses to the EL2 translation tables, are Non-cacheable for all levels of data and unified cache.</td>
</tr>
<tr>
<td>0b1</td>
<td>All data access to Normal memory from EL2, and all accesses to the EL2 translation tables, can be cached at all levels of data and unified cache.</td>
</tr>
</tbody>
</table>

This bit has no effect on the PL1&0 translation regime.

In a system where the PE resets into EL2, this field resets to 0.

A, bit [1]
Alignment check enable. This is the enable bit for Alignment fault checking at EL2:
Meaning

Alignment fault checking disabled when executing at EL2. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element or data elements being accessed.

Alignment fault checking enabled when executing at EL2. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element or data elements being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

In a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 stage 1 address translation. Possible values of this bit are:

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL2 stage 1 address translation disabled. See the HSCTRL.R1 field for the behavior of instruction accesses to Normal memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL2 stage 1 address translation enabled.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2, this field resets to 0.

Accessing the HSCTRL

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>b100</td>
<td>b0001</td>
<td>b0000</td>
<td>b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        return HSCTRL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HSCTRL;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>b100</td>
<td>b0001</td>
<td>b0000</td>
<td>b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
else
    UNDEFINED;
eelif PSTATE.EL == EL2 then
    HSCTRL = R[t];
eelsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSCTRL = R[t];
HSR, Hyp Syndrome Register

The HSR characteristics are:

**Purpose**

Holds syndrome information for an exception taken to Hyp mode.

**Configuration**

AArch32 System register HSR bits [31:0] are architecturally mapped to AArch64 System register ESR_EL2[31:0]. If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HSR is a 32-bit register.

**Field descriptions**

The HSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| EC | IL | ISS |

Execution in any Non-secure PE mode other than Hyp mode makes this register UNKNOWN.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL2, the value of HSR is UNKNOWN. The value written to HSR must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

**EC, bits [31:26]**

Exception Class. Indicates the reason for the exception that this register holds information about. Possible values of this field are:
<table>
<thead>
<tr>
<th>EC</th>
<th>Meaning</th>
<th>ISS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Unknown reason.</td>
<td>ISS encoding for exceptions with an unknown reason</td>
</tr>
<tr>
<td>0b000001</td>
<td>Trapped WFI or WFE instruction execution. Conditional WFE and WFI instructions that fail their condition code check do not cause an exception.</td>
<td>ISS encoding for Exception from a WFI or WFE instruction</td>
</tr>
<tr>
<td>0b000011</td>
<td>Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for Exception from an MCR or MRC access</td>
</tr>
<tr>
<td>0b000100</td>
<td>Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000.</td>
<td>ISS encoding for Exception from an MCRR or MRRC access</td>
</tr>
<tr>
<td>0b000101</td>
<td>Trapped MCR or MRC access with (coproc==0b1110).</td>
<td>ISS encoding for Exception from an MCR or MRC access</td>
</tr>
<tr>
<td>0b000110</td>
<td>Trapped LDC or STC access. The only architectured uses of these instructions are: • An STC to write data to memory from DBGDTRRXint. • An LDC to read data from memory to DBGDTRTXint.</td>
<td>ISS encoding for Exception from an LDC or STC instruction</td>
</tr>
<tr>
<td>0b001000</td>
<td>Access to Advanced SIMD or floating-point functionality trapped by a HCPTR {TASE, TCP10} control. Excludes exceptions generated because Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000.</td>
<td>ISS encoding for Exception from an access to SIMD or floating-point functionality resulting from HCPTR</td>
</tr>
<tr>
<td>0b001000</td>
<td>Trapped VMRS access, from ID group trap, that is not reported using EC 0b000111.</td>
<td>ISS encoding for Exception from an MCR or MRC access</td>
</tr>
<tr>
<td>0b001100</td>
<td>Trapped MRRC access with (coproc==0b1110).</td>
<td>ISS encoding for Exception from an MCRR or MRRC access</td>
</tr>
<tr>
<td>0b001110</td>
<td>Illegal exception return to AArch32 state.</td>
<td>ISS encoding for Exception from an Illegal state or PC alignment fault</td>
</tr>
<tr>
<td>0b010001</td>
<td>Exception on SVC instruction execution in AArch32 state routed to EL2.</td>
<td>ISS encoding for Exception from HVC or SVC instruction execution</td>
</tr>
<tr>
<td>0b010010</td>
<td>HVC instruction execution in AArch32 state, when HVC is not disabled.</td>
<td>ISS encoding for Exception from HVC or SVC instruction execution</td>
</tr>
<tr>
<td>0b010011</td>
<td>Trapped execution of SMC instruction in AArch32 state.</td>
<td>ISS encoding for Exception from SMC instruction execution</td>
</tr>
<tr>
<td>0b100000</td>
<td>Prefetch Abort from a lower Exception level.</td>
<td>ISS encoding for Exception from a Prefetch Abort</td>
</tr>
<tr>
<td>0b100001</td>
<td>Prefetch Abort taken without a change in Exception level.</td>
<td>ISS encoding for Exception from a Prefetch Abort</td>
</tr>
<tr>
<td>0b100010</td>
<td>PC alignment fault exception.</td>
<td>ISS encoding for Exception from an Illegal state or PC alignment fault</td>
</tr>
</tbody>
</table>
Data Abort from a lower Exception level.

Data Abort taken without a change in Exception level.

All other EC values are reserved by Arm, and:

- Unused values in the range `0b000000 - 0b101100` (`0x0 - 0x2C`) are reserved for future use for synchronous exceptions.
- Unused values in the range `0b101101 - 0b111111` (`0x2D - 0x3F`) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is **CONSTRAINED UNPREDICTABLE**, as described in 'Reserved values in System and memory-mapped registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.2.2.

This field resets to an architecturally **UNKNOWN** value.

### IL, bit [25]

Instruction length bit. Indicates the size of the instruction that has been trapped to Hyp mode. When this bit is valid, possible values of this bit are:

<table>
<thead>
<tr>
<th>IL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>16-bit instruction trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>32-bit instruction trapped.</td>
</tr>
</tbody>
</table>

This field is **RES1** and not valid for the following cases:

- When the EC field is `0b000000`, indicating an exception with an unknown reason.
- Prefetch Aborts.
- Data Aborts for which the HSR.ISS.ISV field is 0.
- When the EC value is `0b001110`, indicating an Illegal state exception.

**Note**

This is a change from the behavior in Armv7, where the IL field is UNK/SBZP for the corresponding cases.

The IL field is not valid and is **UNKNOWN** on an exception from a PC alignment fault.

This field resets to an architecturally **UNKNOWN** value.

### ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

#### ISS encoding for exceptions with an unknown reason

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |

**Bits [24:0]**

Reserved, **RES0**.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or is not accessible in the current PE mode in the current Security state, including:
- A read access using a System register encoding pattern that is not allocated for reads or that does not permit reads in the current PE mode and Security state.
- A write access using a System register encoding pattern that is not allocated for writes or that does not permit writes in the current PE mode and Security state.
- Instruction encodings that are unallocated.
- Instruction encodings for instructions not implemented in the implementation.

- In Debug state, the attempted execution of an instruction bit pattern that not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that not accessible in Non-debug state.
- The attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the SCTLR. {ITD, SED, CP15BEN} control bits.
- Attempted execution of:
  - An HVC instruction when disabled by HCR.HCD, SCR.HCE, or SCR_EL3.HCE.
  - An SMC instruction when disabled by SCR.SCD or SCR_EL3.SMD.
  - An HLT instruction when disabled by EDSCR.HDE.
- An exception generated because of the attempted execution of an MSR (Banked register) or MRS (Banked register) instruction that would access a Banked register that is not accessible from the Security state and PE mode at which the instruction was executed.

An exception is generated only if the constrained unpredictable behavior of the instruction is that it is undefined, see 'MSR/MRS Banked registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.29 (Constrained Unpredictable behavior of EL2 features).

- Attempted execution, in Debug state, of:
  - A DCPS1 instruction in Non-secure state from EL0 when EL2 is using AArch32 and the value of HCR.TGE is 1.
  - A DCPS2 instruction at EL1 or EL0 when EL2 is not implemented, or when EL3 is using AArch32 and the value of SCR.NS is 0, or when EL3 is using AArch64 and the value of SCR_EL3.NS is 0.
  - A DCPS3 instruction when EL3 is not implemented, or when the value of EDSCR.SDD is 1.
- In Debug state when the value of EDSCR.SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.

'Undefined Instruction exception, when HCR.TGE is set to 1' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers’ Model), describes the configuration settings for a trap that returns an HSR.EC value of 0b000000.

**ISS encoding for Exception from a WFI or WFE instruction**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | RES0 | TI |

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.
**COND, bits [23:20]**

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to \(0b1110\).

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to \(0b1110\), the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether:

- CV is set to 0 and COND is set to an **UNKNOWN** value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is **IMPLEMENTATION DEFINED** whether the COND field is set to \(0b1110\), or to the value of any condition that applied to the instruction.

This field resets to an architecturally **UNKNOWN** value.

**Bits [19:1]**

Reserved, RES0.

**TI, bit [0]**

Trapped instruction. Possible values of this bit are:

<table>
<thead>
<tr>
<th>TI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>WFI trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>WFE trapped.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

'Trapping use of the WFI and WFE instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers' Model), describes the configuration settings for this trap.

**ISS encoding for Exception from an MCR or MRC access**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally **UNKNOWN** value.
**COND, bits [23:20]**

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to $0b1110$.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to $0b1110$, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is implementation defined whether:

- CV is set to 0 and COND is set to an unknown value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is implementation defined whether the COND field is set to $0b1110$, or to the value of any condition that applied to the instruction.

This field resets to an architecturally unknown value.

**Opc2, bits [19:17]**

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value $0b000$.

This field resets to an architecturally unknown value.

**Opc1, bits [16:14]**

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value $0b111$.

This field resets to an architecturally unknown value.

**CRn, bits [13:10]**

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

This field resets to an architecturally unknown value.

**Bit [9]**

Reserved, RES0.

**Rt, bits [8:5]**

The Rt value from the issued instruction, the general-purpose register used for the transfer.

This field resets to an architecturally unknown value.

**CRm, bits [4:1]**

The CRm value from the issued instruction.
For a trapped VMRS access, holds the value \(0b0000\).

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0)</td>
<td>Write to System register space. MCR instruction.</td>
</tr>
<tr>
<td>(0b1)</td>
<td>Read from System register space. MRC or VMRS instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following sections describe configuration settings for traps that are reported using EC value \(0b000011\):

- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to the ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers’ Model).
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to lockdown, DMA, and TCM operations' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL1 accesses to the Auxiliary Control Register' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL1 accesses to the CPACR' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL1 accesses to virtual memory control registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'General trapping to Hyp mode of Non-secure EL0 and EL1 accesses to System registers in the (coproc == 1111) encoding space' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.

The following sections describe configuration settings for traps that are reported using EC value \(0b000101\):

- 'ID group 0, Primary device identification registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.

The following sections describes configuration settings for traps that are reported using EC value \(0b010000\):

- 'ID group 0, Primary device identification registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.

### ISS encoding for Exception from an MCRR or MRRC access

<table>
<thead>
<tr>
<th>CV</th>
<th>COND</th>
<th>Opc1</th>
<th>RES0</th>
<th>Rt2</th>
<th>RES0</th>
<th>Rt</th>
<th>CRm</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
</tr>
</tbody>
</table>
**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally UNKNOWN value.

**COND, bits [23:20]**

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Opc1, bits [19:16]**

The Opc1 value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

**Bits [15:14]**

Reserved, RES0.

**Rt2, bits [13:10]**

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

This field resets to an architecturally UNKNOWN value.

**Bit [9]**

Reserved, RES0.

**Rt, bits [8:5]**

The Rt value from the issued instruction, the first general-purpose register used for the transfer.
This field resets to an architecturally **UNKNOWN** value.

**CRm, bits [4:1]**

The CRm value from the issued instruction.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to System register space. MCRR instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from System register space. MRRC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

The following sections describe configuration settings for traps that are reported using EC value 0b000100:

- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Performance Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Activity Monitors registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to the Generic Timer registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.
- 'General trapping to Hyp mode of Non-secure EL0 and EL1 accesses to System registers in the (coproc == 1111) encoding space' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1.

The following sections describe configuration settings for traps that are reported using EC value 0b001100:


**ISS encoding for Exception from an LDC or STC instruction**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CV | COND | imm8 | RES0 | Rn | Offset | AM | Direction |

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally **UNKNOWN** value.

**COND, bits [23:20]**

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:
If the instruction is conditional, COND is set to the condition code field value from the instruction.
If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**imm8, bits [19:12]**

The immediate value from the issued instruction.

This field resets to an architecturally UNKNOWN value.

**Bits [11:9]**

Reserved, RES0.

**Rn, bits [8:5]**

The Rn value from the issued instruction. Valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction.

When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**Offset, bit [4]**

Indicates whether the offset is added or subtracted:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Subtract offset.</td>
</tr>
<tr>
<td>0b1</td>
<td>Add offset.</td>
</tr>
</tbody>
</table>

This bit corresponds to the U bit in the instruction encoding.

This field resets to an architecturally UNKNOWN value.

**AM, bits [3:1]**

Addressing mode. The permitted values of this field are:
The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is **CONSTRAINED UNPREDICTABLE**, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits \{P, W\} in the instruction encoding.

This field resets to an architecturally **UNKNOWN** value.

**Direction, bit [0]**

Indicates the direction of the trapped instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Direction</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Write to memory. STC instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read from memory. LDC instruction.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

'Trapping general Non-secure System register accesses to debug registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 describes the configuration settings for the trap that is reported using EC value 0b000110.

**ISS encoding for Exception from an access to SIMD or floating-point functionality, resulting from HCPTR**

```
24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
| CV | COND | RES0 | TA RES0 | coproc |
```

Excludes exceptions that occur because Advanced SIMD and floating-point functionality is not implemented, or because the value of \(HCR\).TGE or \(HCR\_EL2\).TGE is 1. These are reported with EC value 0b000000.

**CV, bit [24]**

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is **IMPLEMENTATION DEFINED** whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field resets to an architecturally **UNKNOWN** value.

**COND, bits [23:20]**

The condition code for the trapped instruction.
When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to $0b1110$.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to $0b1110$, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to $0b1110$, or to the value of any condition that applied to the instruction.

This field resets to an architecturally UNKNOWN value.

**Bits [19:6]**

Reserved, RES0.

**TA, bit [5]**

Indicates trapped use of Advanced SIMD functionality. The possible values of this bit are:

<table>
<thead>
<tr>
<th>TA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0b0$</td>
<td>Exception was not caused by trapped use of Advanced SIMD functionality.</td>
</tr>
<tr>
<td>$0b1$</td>
<td>Exception was caused by trapped use of Advanced SIMD functionality.</td>
</tr>
</tbody>
</table>

Any use of an Advanced SIMD instruction that is not also a floating-point instruction that is trapped to Hyp mode because of a trap configured in the HCPTR sets this bit to 1.

For a list of these instructions, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.

This field resets to an architecturally UNKNOWN value.

**Bit [4]**

Reserved, RES0.

**coproc, bits [3:0]**

When the TA field returns the value 1, this field returns the value 1010, otherwise this field is RES0.

This field resets to an architecturally UNKNOWN value.

The following sections describe the configuration settings for the traps that are reported using EC value $0b001111$:


**ISS encoding for Exception from HVC or SVC instruction execution**
Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, this is the value of the imm16 field of the issued instruction.

For an SVC instruction:

- If the instruction is unconditional, then:
  - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
  - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

The HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

‘Supervisor Call exception, when HCR.TGE is set to 1’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers’ Model), describes the configuration settings for the trap reported with EC value 0b010001.

ISS encoding for Exception from SMC instruction execution

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | CCKNOWLEDGEPASS | RES0 |

CV, bit [24]

Condition code valid. Possible values of this bit are:

<table>
<thead>
<tr>
<th>CV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The COND field is not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The COND field is valid.</td>
</tr>
</tbody>
</table>

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is only valid if CCKNOWLEDGEPASS is 1, otherwise it is RES0.

This field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:
• With COND set to 0b1110, the value for unconditional.
• With the COND value held in the instruction.

When a T32 instruction is trapped, it is implementation defined whether:

• CV is set to 0 and COND is set to an unknown value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
• CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is implementation defined whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is only valid if CCKNOWNPASS is 1, otherwise it is res0.

This field resets to an architecturally unknown value.

**CCKNOWNPASS, bit [19]**

Indicates whether the instruction might have failed its condition code check.

<table>
<thead>
<tr>
<th>CCKNOWNPASS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The instruction was unconditional, or was conditional and passed its condition code check.</td>
</tr>
<tr>
<td>0b1</td>
<td>The instruction was conditional, and might have failed its condition code check.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally unknown value.

**Bits [18:0]**

Reserved, res0.

‘Traps to Hyp mode of Non-secure EL1 execution of SMC instructions’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers’ Model), describes the configuration settings for this trap, for instructions executed in Non-secure EL1.

**ISS encoding for Exception from a Prefetch Abort**

| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|---|---|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 | FnVEA | RES0S1PTW | RES0 | IFSC |

**Bits [24:11]**

Reserved, res0.

**FnV, bit [10]**

FAR not valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>HIFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>HIFAR is not valid, and holds an unknown value.</td>
</tr>
</tbody>
</table>

This field is only valid if the IFSC code is 0b1000000. It is res0 for all other aborts.

This field resets to an architecturally unknown value.

**EA, bit [9]**

External abort type. This bit can provide an implementation defined classification of External aborts.
For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally **UNKNOWN** value.

**Bit [8]**

Reserved, RES0.

**S1PTW, bit [7]**

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>

For any abort other than a stage 2 fault this bit is RES0.

This field resets to an architecturally **UNKNOWN** value.

**Bit [6]**

Reserved, RES0.

**IFSC, bits [5:0]**

Instruction Fault Status Code. Possible values of this field are:

<table>
<thead>
<tr>
<th>IFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Address size fault, translation table base register.</td>
</tr>
<tr>
<td>0b00001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b00011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b00101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b00110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b00111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b01000</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b01001</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b01010</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b01100</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b01101</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b01110</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b01111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100010</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
</tbody>
</table>

All other values are reserved.
When the RAS Extension is implemented, 0b011000, 0b011101, 0b011110, and 0b011111, are reserved.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-selector translation table lookup' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally **UNKNOWN** value.

The following sections describe cases where Prefetch Abort exceptions can be routed to Hyp mode, generating exceptions that are reported in the HSR with EC value 0b100000:

- ‘Abort exceptions, when HCR.TGE is set to 1’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G1 (The AArch32 System Level Programmers' Model).

### ISS encoding for Exception from an Illegal state or PC alignment fault

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

For more information about the Illegal state exception, see:


For more information about the PC alignment fault exception, see 'Branching to an unaligned PC' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, appendix A.

### ISS encoding for Exception from a Data Abort

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**ISV, bit [24]**

Instruction syndrome valid. Indicates whether the syndrome information in ISS[23:14] is valid.

<table>
<thead>
<tr>
<th>ISV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No valid instruction syndrome. ISS[23:14] are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>ISS[23:14] hold a valid instruction syndrome.</td>
</tr>
</tbody>
</table>

This bit is 0 for all faults except Data Aborts generated by stage 2 address translations for which all the following apply to the instruction that generated the Data Abort exception:

- The instruction is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STL, or STRBT instruction.
• The instruction is not performing register writeback.
• The instruction is not using the PC as a source or destination register.

For these cases, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, as described in 'Data Aborts in Memory access mode' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H4.3.2 (Memory access mode), and otherwise indicates whether ISS[23:14] hold a valid syndrome.

**Note**

In the A32 instruction set, LDR*T and STR*T instructions always perform register writeback and therefore never return a valid instruction syndrome.

When the RAS Extension is implemented, ISV is 0 for any synchronous External abort.

ISV is set to 0 on a stage 2 abort on a stage 1 translation table walk.

When the RAS Extension is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

This field resets to an architecturally UNKNOWN value.

**SAS, bits [23:22]**

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

<table>
<thead>
<tr>
<th>SAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Byte</td>
</tr>
<tr>
<td>0b01</td>
<td>Halfword</td>
</tr>
<tr>
<td>0b10</td>
<td>Word</td>
</tr>
<tr>
<td>0b11</td>
<td>Doubleword</td>
</tr>
</tbody>
</table>

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

**SSE, bit [21]**

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

<table>
<thead>
<tr>
<th>SSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sign-extension not required.</td>
</tr>
<tr>
<td>0b1</td>
<td>Data item must be sign-extended.</td>
</tr>
</tbody>
</table>

For all other operations this bit is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

This field resets to an architecturally UNKNOWN value.

**Bit [20]**

Reserved, RES0.

**SRT, bits [19:16]**

Syndrome Register transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction.

This field is UNKNOWN when the value of ISV is UNKNOWN.
This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**Bit [15]**

Reserved, **RES0**.

**AR, bit [14]**

Acquire/Release. When ISV is 1, the possible values of this bit are:

<table>
<thead>
<tr>
<th>AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Instruction did not have acquire/release semantics.</td>
</tr>
<tr>
<td>0b1</td>
<td>Instruction did have acquire/release semantics.</td>
</tr>
</tbody>
</table>

This field is **UNKNOWN** when the value of ISV is **UNKNOWN**.

This field is **RES0** when the value of ISV is 0.

This field resets to an architecturally **UNKNOWN** value.

**Bits [13:12]**

Reserved, **RES0**.

**AET, bits [11:10]**

**When RAS is implemented:**

Asynchronous Error Type.

When the RAS Extension is implemented and the value returned in the DFSC field is 0b010001, describes the state of the PE after taking the SError interrupt exception. The possible values of this field are:

<table>
<thead>
<tr>
<th>AET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Uncontainable error (UC) or uncategorized.</td>
</tr>
<tr>
<td>0b01</td>
<td>Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b10</td>
<td>Restartable error (UEO) or Corrected error (CE).</td>
</tr>
<tr>
<td>0b11</td>
<td>Recoverable error (UER).</td>
</tr>
</tbody>
</table>

On a synchronous Data Abort, this field is **RES0**.

If multiple errors are taken as a single SError interrupt exception, the overall state of the PE is reported. For example, if both a Recoverable and Unrecoverable error occurred, the state is Unrecoverable.

**Note**

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

When the RAS Extension is not implemented, or when DFSC is not 0b010001:

- Bit[11] is **RES0**.
- Bit[10] forms the FnV field.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>HDFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>HDFAR is not valid, and holds an \textit{UNKNOWN} value.</td>
</tr>
</tbody>
</table>

When the RAS Extension is not implemented, this field is valid only if DFSC is 0b010000. It is \textit{RES}0 for all other aborts.

When the RAS Extension is implemented:

- If DFSC is 0b010000, this field is valid.
- If DFSC is 0b010001, this bit forms part of the AET field, becoming AET[0].
- This field is \textit{RES}0 for all other aborts.

\textbf{Note}

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{EA, bit [9]}

External abort type. This bit can provide an \textit{IMPLEMENTATION DEFINED} classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{CM, bit [8]}

Cache maintenance. For a synchronous fault, identifies fault that comes from a cache maintenance or address translation instruction. For synchronous faults, the possible values of this bit are:

<table>
<thead>
<tr>
<th>CM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not generated by a cache maintenance or address translation instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault generated by a cache maintenance or address translation instruction.</td>
</tr>
</tbody>
</table>

For an asynchronous Data Abort exception, this bit is 0.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{S1PTW, bit [7]}

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

<table>
<thead>
<tr>
<th>S1PTW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault not on a stage 2 translation for a stage 1 translation table walk.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault on the stage 2 translation of an access for a stage 1 translation table walk.</td>
</tr>
</tbody>
</table>

For any abort other than a stage 2 fault this bit is \textit{RES}0.

This field resets to an architecturally \textit{UNKNOWN} value.
**WnR, bit [6]**

Write not Read. Indicates whether a synchronous abort was caused by a write instruction or a read instruction. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WnR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Abort caused by a read instruction.</td>
</tr>
<tr>
<td>0b1</td>
<td>Abort caused by a write instruction.</td>
</tr>
</tbody>
</table>

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

On an asynchronous Data Abort:

- When the RAS Extension is not implemented, this bit is **UNKNOWN**.
- When the RAS Extension is implemented, this bit is **RES0**.

**Note**

Armv8.2 requires the implementation of the RAS Extension.

This field resets to an architecturally **UNKNOWN** value.

**DFSC, bits [5:0]**

Data Fault Status Code. Possible values of this field are:

<table>
<thead>
<tr>
<th>DFSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Address size fault, translation table base register.</td>
</tr>
<tr>
<td>0b00001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b00011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b00101</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b00110</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b00111</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b01001</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b01010</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b01011</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b01101</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b01110</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b01111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b10000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b10001</td>
<td><strong>S</strong>Error interrupt.</td>
</tr>
<tr>
<td>0b10101</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b10110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b11000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b11001</td>
<td><strong>S</strong>Error interrupt from a parity or ECC error on memory access.</td>
</tr>
<tr>
<td>0b11101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b11110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b11111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b10000</td>
<td>Alignment fault.</td>
</tr>
<tr>
<td>0b10001</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>0b11000</td>
<td>TLB conflict abort.</td>
</tr>
<tr>
<td>0b11100</td>
<td><strong>IMPLEMENTATION</strong> DEFINED fault (Lockdown).</td>
</tr>
<tr>
<td>0b11101</td>
<td><strong>IMPLEMENTATION</strong> DEFINED fault (Unsupported Exclusive access).</td>
</tr>
</tbody>
</table>

All other values are reserved.
When the RAS Extension is implemented, \(0b011000, 0b011001, 0b011101, 0b011110,\) and \(0b011111,\) are reserved.

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults on a Long-descriptor translation table lookup’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

This field resets to an architecturally **UNDEFINED** value.

The following describe cases where Data Abort exceptions can be routed to Hyp mode, generating exceptions that are reported in the HSR with EC value \(0b100100\):


The following describe cases that can cause a Data Abort exception that is taken to Hyp mode, and reported in the HSR with EC value of \(0b100000\) or \(0b100100\):


### Accessing the HSR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>proc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

\[
\text{if PSTATE.EL == EL0 then}
\text{UNDEFINED;}
\text{elsif PSTATE.EL == EL1 then}
\text{if EL2Enabled() \&\& !ELUsingAArch32(EL2) \&\& HSTR_EL2.T5 == '1' then}
\text{AArch64.AArch32SystemAccessTrap(EL2, 0x03);}
\text{elsif EL2Enabled() \&\& ELUsingAArch32(EL2) \&\& HSTR.T5 == '1' then}
\text{AArch32.TakeHypTrapException(0x03);}
\text{else}
\text{UNDEFINED;}
\text{elsif PSTATE.EL == EL2 then}
\text{return HSR;}
\text{elsif PSTATE.EL == EL3 then}
\text{if SCR.NS == '0' then}
\text{UNDEFINED;}
\text{else}
\text{return HSR;}
\]

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>proc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSR = R[t];
HSTR, Hyp System Trap Register

The HSTR characteristics are:

**Purpose**

Controls trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to System registers in the coproc == 0b1111 encoding space:

- By the CRn value used to access the register using MCR or MRC instruction.
- By the CRm value used to access the register using MCRR or MRRC instruction.

**Configuration**

AArch32 System register HSTR bits [31:0] are architecturally mapped to AArch64 System register HSTR_EL2[31:0]. If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HSTR is a 32-bit register.

**Field descriptions**

The HSTR bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| RES0 | T15 | T14 | T13 | T12 | T11 | T10 | T9 | T8 | T7 | T6 | T5 | T4 | T3 | T2 | T1 | T0 |

**Bits [31:16]**

Reserved, RES0.

**T<n>, bit [n], for n = 0 to 15**

Fields T14 and T4 are RES0.

The remaining fields control whether Non-secure EL0 and EL1 accesses, using MCR, MRC, MCRR, and MRRC instructions, to the System registers in the coproc == 0b1111 encoding space are trapped to Hyp mode:

<table>
<thead>
<tr>
<th>T&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on Non-secure EL0 or EL1 accesses to System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any Non-secure EL1 MCR or MRC access with coproc == 0b1111 and CRn == &lt;n&gt; is trapped to Hyp mode. A Non-secure EL0 MCR or MRC access with these values is trapped to Hyp mode only if the access is not UNDEFINED when the value of this field is 0. Any Non-secure EL1 MCRR or MRRC access with coproc == 0b1111 and CRm == &lt;n&gt; is trapped to Hyp mode. A Non-secure EL0 MCRR or MRRC access with these values is trapped to Hyp mode only if the access is not UNDEFINED when the value of this field is 0.</td>
</tr>
</tbody>
</table>

For example, when HSTR.T7 is 1, for instructions executed at Non-secure EL1:
• An MCR or MRC instruction with coproc set to 0b1111 and CRn set to c7 is trapped to Hyp mode.
• An MCRR or MRRC instruction with coproc set to 0b1111 and CRm set to c7 is trapped to Hyp mode.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**Accessing the HSTR**

Accesses to this register use the following encodings:

**MRC{c}{q} {<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>}, {#}<opc2>**

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
 UNDEFINED;
elif PSTATE.EL == EL1 then
 if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
  AArch32.TakeHypTrapException(0x03);
else
  UNDEFINED;
elsif PSTATE.EL == EL2 then
 return HSTR;
elif PSTATE.EL == EL3 then
 if SCR.NS == '0' then
  UNDEFINED;
else
  return HSTR;

**MCR{c}{q} {<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>}, {#}<opc2>**

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
 UNDEFINED;
elif PSTATE.EL == EL1 then
 if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
  AArch32.TakeHypTrapException(0x03);
else
  UNDEFINED;
elsif PSTATE.EL == EL2 then
 HSTR = R[t];
elif PSTATE.EL == EL3 then
 if SCR.NS == '0' then
  UNDEFINED;
else
  HSTR = R[t];
HTCR, Hyp Translation Control Register

The HTCR characteristics are:

**Purpose**

The control register for stage 1 of the EL2 translation regime.

**Note**

This stage of translation always uses the Long-descriptor translation table format.

**Configuration**

AArch32 System register HTCR bits [31:0] are architecturally mapped to AArch64 System register TCR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HTCR is a 32-bit register.

**Field descriptions**

The HTCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES1.</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED, bit [30]</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>28</td>
<td>This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>HWU62, bit [28]</td>
</tr>
<tr>
<td>25</td>
<td>When ARMv8.2-TTPBHA is implemented:</td>
</tr>
<tr>
<td>24</td>
<td>Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.</td>
</tr>
</tbody>
</table>
# HTCR, Hyp Translation Control Register

<table>
<thead>
<tr>
<th>HWU62</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

## HWU61, bit [27]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU61</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

## HWU60, bit [26]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU60</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

## HWU59, bit [25]
When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU59</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD, bit [24]

When ARMv8.2-AA32HPD is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the PL2 translation regime.

<table>
<thead>
<tr>
<th>HPD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

Bits [22:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using HTTBR.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to an architecturally UNKNOWN value.
**ORGN0, bits [11:10]**

Outer cacheability attribute for memory associated with translation table walks using **HTTBR**.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**IRGN0, bits [9:8]**

Inner cacheability attribute for memory associated with translation table walks using **HTTBR**.

<table>
<thead>
<tr>
<th>IRGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [7:3]**

Reserved, RES0.

**T0SZ, bits [2:0]**

The size offset of the memory region addressed by **HTTBR**. The region size is $2^{(32-T0SZ)}$ bytes.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the HTCR**

Accesses to this register use the following encodings:

```
MRC{<c>{<q>}} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HTCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HTCR;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTCR = R[t];
else
    UNDEFINED;
HTPIDR, Hyp Software Thread ID Register

The HTPIDR characteristics are:

**Purpose**

Provides a location where software running in Hyp mode can store thread identifying information that is not visible to Non-secure software executing at EL0 or EL1, for hypervisor management purposes.

The PE makes no use of this register.

**Configuration**

AArch32 System register HTPIDR bits [31:0] are architecturally mapped to AArch64 System register TPIDR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

---

**Note**

The PE never updates this register.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HTPIDR is a 32-bit register.

**Field descriptions**

The HTPIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Thread ID. Thread identifying information stored by software running at this Exception level.

This field resets to an architecturally UNKNOWN value.

**Accessing the HTPIDR**

Accesses to this register use the following encodings:

$$\text{MRC}\{<c><q>\} \langle\text{coproc}\rangle, \{#\}<\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle\{, \{#\}<\text{opc2}\rangle\}$$

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return HTPIDR;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    return HTPIDR;
MCR{c}{q} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  HTPIDR = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HTPIDR = R[t];
HTRFCR, Hyp Trace Filter Control Register

The HTRFCR characteristics are:

**Purpose**

Provides EL2 controls for Trace.

**Configuration**

AArch32 System register HTRFCR bits [31:0] are architecturally mapped to AArch64 System register TRFCR_EL2[31:0].

This register is present only when ARMv8.4-Trace is implemented. Otherwise, direct accesses to HTRFCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from Monitor mode when SCR.NS == 1.

This register is in the Warm reset domain. Some or all RW fields of this register have defined reset values. On a Warm or Cold reset these apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, on a Warm or Cold reset RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HTRFCR is a 32-bit register.

**Field descriptions**

The HTRFCR bit assignments are:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|--------------------------------|---------------------------------|---------------------------------|
| RES0                          | TS                              | RES0CXRES0E2TRE                  |

**Bits [31:7]**

Reserved, RES0.

**TS, bits [6:5]**

Timestamp Control. Controls which timebase is used for trace timestamps.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The timestamp is controlled by TRFCR.TS.</td>
</tr>
<tr>
<td>0b01</td>
<td>Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF.</td>
</tr>
<tr>
<td>0b11</td>
<td>Physical timestamp. The traced timestamp is the physical counter value.</td>
</tr>
</tbody>
</table>

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Bit [4]**

Reserved, RES0.
**CX, bit [3]**

VMID Trace Enable.

<table>
<thead>
<tr>
<th>CX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>VMID tracing is not allowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>VMID tracing is allowed.</td>
</tr>
</tbody>
</table>

When `SelfHostedTraceEnabled() == FALSE`, this field is ignored.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Bit [2]**

Reserved, RES0.

**E2TRE, bit [1]**

EL2 Trace Enable.

<table>
<thead>
<tr>
<th>E2TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing is prohibited at EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing is allowed at EL2.</td>
</tr>
</tbody>
</table>

When `SelfHostedTraceEnabled() == FALSE`, this field is ignored.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**E0HTRE, bit [0]**

EL0 Trace Enable.

<table>
<thead>
<tr>
<th>E0HTRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing is prohibited at EL0 when <code>HCR.TGE == 1</code>.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing is allowed at EL0 when <code>HCR.TGE == 1</code>.</td>
</tr>
</tbody>
</table>

This field is ignored if any of the following are true:

- The PE is in Secure state.
- `SelfHostedTraceEnabled() == FALSE`.
- `HCR.TGE == 0`.

On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

**Accessing the HTRFCR**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>{, \{#<opc2>\}\}}} 
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return HTRFCR;
elseif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
else
    return HTRFCR;

MCR{<c>}{<q>}{<coproc>}, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elseif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
    AArch32.TakeMonitorTrapException();
  else
    HTRFCR = R[t];
elseif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
else
    HTRFCR = R[t];
The HTTBR characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

**Configuration**

AArch32 System register HTTBR bits [47:1] are architecturally mapped to AArch64 System register TTBR0_EL2[47:1].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HTTBR is a 64-bit register.

**Field descriptions**

The HTTBR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BADDR | BADDR |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:48]**

Reserved, RES0.

**BADDR, bits [47:1]**

Translation table base address, bits[47:x]. Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of HTCR.T0SZ as follows:

- If HTCR.T0SZ is 0 or 1, x = 5 - HTCR.T0SZ.
- If HTCR.T0SZ is greater than 1, x = 14 - HTCR.T0SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

This field resets to an architecturally UNKNOWN value.
CnP, bit [0]

When ARMv8.2-TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by HTTBR is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of HTTBR.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by HTTBR are permitted to differ from corresponding entries for HTTBR for other PEs in the Inner Shareable domain. This is not affected by the value of HTTBR.CnP on those other PEs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by HTTBR are the same as the translation table entries pointed to by HTTBR on every other PE in the Inner Shareable domain for which the value of HTTBR.CnP is 1.</td>
</tr>
</tbody>
</table>

**Note**

If the value of the HTTBR.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those HTTBRs do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Accessing the HTTBR**

Accesses to this register use the following encodings:

\[
\text{MRRC}\{<c>\}\{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{Rt2}>, <\text{CRm}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
     AArch64.AArch32SystemAccessTrap(EL2, 0x04);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
     AArch32.TakeHypTrapException(0x04);
   else
     UNDEFINED;
   endif
elsif PSTATE.EL == EL2 then
   return HTTBR;
elsif PSTATE.EL == EL3 then
   if SCR.NS == '0' then
     UNDEFINED;
   else
     return HTTBR;
   endif
else
   return HTTBR;
MCRR{<c>}{<q>}, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
    AArch32.TakeHypTrapException(0x04);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  HTTBR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    HTTBR = R[t2]:R[t];

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
HVBAR, Hyp Vector Base Address Register

The HVBAR characteristics are:

**Purpose**

Holds the vector base address for any exception that is taken to Hyp mode.

**Configuration**

AArch32 System register HVBAR bits [31:0] are architecturally mapped to AArch64 System register VBAR_EL2[31:0]. If EL2 is not implemented, this register is RES0 from EL3. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

HVBAR is a 32-bit register.

**Field descriptions**

The HVBAR bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| **Vector Base Address** | **RES0** |

**Bits [31:5]**

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

This field resets to an architecturally UNKNOWN value.

**Bits [4:0]**

Reserved, RES0.

**Accessing the HVBAR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

Page 2562
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HVBAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HVBAR;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HVBAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HVBAR = R[t];
ICC_AP0R<n>, Interrupt Controller Active Priorities
Group 0 Registers, n = 0 - 3

The ICC_AP0R<n> characteristics are:

**Purpose**

Provides information about Group 0 active priorities.

**Configuration**

AArch32 System register ICC_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register
ICC_AP0R<n>_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception
level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICC_AP0R<n> is a 32-bit register.

**Field descriptions**

The ICC_AP0R<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**IMPLEMENTATION DEFINED, bits [31:0]**

**IMPLEMENTATION DEFINED.**

This field resets to 0.

The contents of these registers are **IMPLEMENTATION DEFINED** with the one architectural requirement that the value
0x00000000 is consistent with no interrupts being active.

**Accessing the ICC_AP0R<n>**

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are
no Group 0 active priorities) might result in **UNPREDICTABLE** behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP0R1 is only implemented in implementations that support 6 or more bits of preemption. ICC_AP0R2 and
ICC_AP0R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are
**UNDEFINED**.

**Note**

The number of bits of preemption is indicated by **ICH_VTR.PREbits**.

Writing to the active priority registers in any order other than the following order will result in **UNPREDICTABLE**
behavior:

- ICC_AP0R<n>. 
Secure ICC_AP1R
Non-secure ICC_AP1R

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
</tr>
</tbody>
</table>

If \( \text{PSTATE.EL} == \text{EL0} \) then
UNDEFINED;

elsif \( \text{PSTATE.EL} == \text{EL1} \) then
    if \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR_EL2.T12} == '1' \) then
        \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
    elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR_EL2.T12} == '1' \) then
        \text{AArch32.TakeHypTrapException(0x03)};
    elsif \( \text{ICC_SRE.SRE} == '0' \) then
        UNDEFINED;
    elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{ICH_HCR_EL2.TALL0} == '1' \) then
        \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
    elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{ICH_HCR.TALL0} == '1' \) then
        \text{AArch32.TakeHypTrapException(0x03)};
    elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR_EL2.FMO} == '1' \) then
        return \text{ICV_AP0R[UInt(opc2<1:0>)]};
    elsif \( \text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR.FMO} == '1' \) then
        return \text{ICV_AP0R[UInt(opc2<1:0>)]};
    elsif \( \text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR_EL3.FIQ} == '1' \) then
        \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
    elsif \( \text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR.FIQ} == '1' \) then
        \text{AArch32.TakeMonitorTrapException();}
    else
        return \text{ICV_AP0R[UInt(opc2<1:0>)]};
    endif
else
    return \text{ICV_AP0R[UInt(opc2<1:0>)]};
endsel

elsif \( \text{PSTATE.EL} == \text{EL2} \) then
    if \( \text{ICC_HSRE.SRE} == '0' \) then
        UNDEFINED;
    elsif \( \text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR_EL3.FIQ} == '1' \) then
        \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
    elsif \( \text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR.FIQ} == '1' \) then
        \text{AArch32.TakeMonitorTrapException();}
    else
        return \text{ICV_AP0R[UInt(opc2<1:0>)]};
    endif
else
    return \text{ICV_AP0R[UInt(opc2<1:0>)]};
endsel

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
elif ICC_SRE.SRE == '0' then
    UNDEFINED;
elif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
elif ICC_HSRE.SRE == '0' then
    UNDEFINED;
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
elif ICC_MSRE.SRE == '0' then
    UNDEFINED;
elif ICC_MSRE.SRE == '0' then
        ICC_AP0R[UIntopc2<1:0>] = R[t];
elif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
elif ICC_SRE.SRE == '0' then
    UNDEFINED;
elif ICC_MSRE.SRE == '0' then
    UNDEFINED;
elif ICC_MSRE.SRE == '0' then
        ICC_AP0R[UIntopc2<1:0>] = R[t];
ICC_AP1R<n>, Interrupt Controller Active Priorities Group 1 Registers, n = 0 - 3

The ICC_AP1R<n> characteristics are:

### Purpose

Provides information about Group 1 active priorities.

### Configuration

AArch32 System register ICC_AP1R<n> bits [31:0] (S) are architecturally mapped to AArch64 System register ICC_AP1R<n>_EL1[31:0] (S).

AArch32 System register ICC_AP1R<n> bits [31:0] (NS) are architecturally mapped to AArch64 System register ICC_AP1R<n>_EL1[31:0] (NS).

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

### Attributes

ICC_AP1R<n> is a 32-bit register.

### Field descriptions

The ICC_AP1R<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

### Accessing the ICC_AP1R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP1R1 is only implemented in implementations that support 6 or more bits of preemption. ICC_AP1R2 and ICC_AP1R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

### Note

The number of bits of preemption is indicated by ICH_VTR.PREbits.
Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- **ICC_AP0R<n>**
- Secure ICC_AP1R<n>
- Non-secure ICC_AP1R<n>

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_AP1R[UInt(opc2<1:0>)];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    return ICV_AP1R[UInt(opc2<1:0>)];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    if SCR.NS == '0' then
      return ICC_AP1R_NS[UInt(opc2<1:0>)];
    else
      return ICC_AP1R_S[UInt(opc2<1:0>)];
    end if;
  else
    return ICC_AP1R[UInt(opc2<1:0>)];
  end if;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    return ICC_AP1R_NS[UInt(opc2<1:0>)];
  else
    return ICC_AP1R[UInt(opc2<1:0>)];
  end if;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  elsif SCR.NS == '0' then
    return ICC_AP1R_S[UInt(opc2<1:0>)];
  else
    return ICC_AP1R_NS[UInt(opc2<1:0>)];
  end if;
end if;
```

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && ICC_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && ICC_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif ICC_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif ICC_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ICC_AP1R[UInt(opc2<1:0>)] = R[t];
    end if
    if SCR.NS == '0' then
        ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
    else
        ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
    end if
else
    ICC_AP1R[UInt(opc2<1:0>)] = R[t];
end if
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elseif ICC_HSRE.SRE == '1' then
        if SCR.NS == '0' then
            ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
        else
            ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
        end if
else
    ICC_AP1R[UInt(opc2<1:0>)] = R[t];
end if
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    elseif ICC_MSRE.SRE == '1' then
        if SCR.NS == '0' then
            ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
        else
            ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
        end if
else
    ICC_AP1R[UInt(opc2<1:0>)] = R[t];
end if
The ICC_ASGI1R characteristics are:

**Purpose**

Generates Group 1 SGIs for the Security state that is not the current Security state.

**Configuration**

AArch32 System register ICC_ASGI1R performs the same function as AArch64 System register ICC_ASGI1R_EL1.

Under certain conditions a write to ICC_ASGI1R can generate Group 0 interrupts, see Forwarding an SGI to a target PE.

**Attributes**

ICC_ASGI1R is a 64-bit register.

**Field descriptions**

The ICC_ASGI1R bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | **Aff3** | RS | RES0 | **IRM** | **Aff2** | **TargetList** |
| RES0 | INTID | Aff1 | | | | |

**Bits [63:56]**

Reserved, RES0.

**Aff3, bits [55:48]**

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**RS, bits [47:44]**

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a constrained unpredictable choice of:

- The write is ignored.
- The RS field is treated as 0.
Bits [43:41]  
Reserved, RES0.

IRM, bit [40]  
Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

<table>
<thead>
<tr>
<th>IRM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.&lt;target list&gt;.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to all PEs in the system, excluding &quot;self&quot;.</td>
</tr>
</tbody>
</table>

Aff2, bits [39:32]  
The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]  
Reserved, RES0.

INTID, bits [27:24]  
The INTID of the SGI.

Aff1, bits [23:16]  
The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]  
Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is **IMPLEMENTATION DEFINED** whether, in such cases, a Distributor can signal a system error.

**Note**  
This restricts a system to sending targeted SGIs to PEs with an affinity 0 number that is less than 16. If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

**Accessing the ICC_ASGI1R**  
This register allows software executing in a Secure state to generate Non-secure Group 1 SGIs. It will also allow software executing in a Non-secure state to generate Secure Group 1 SGIs, if permitted by the settings of **GICR_NSACR** in the Redistributor corresponding to the target PE.
When \( \text{GICD_CTLR.DS}==0 \), non-secure writes do not generate an interrupt for a target PE if not permitted by the \( \text{GICR_NSACR} \) register associated with the target PE. For more information see Use of control registers for SGI forwarding.

**Note**

Accesses from Secure Monitor mode are treated as Secure regardless of the value of \( \text{SCR.NS} \).

Accesses to this register use the following encodings:

\[
\text{MCRR\{<c>\}{<q>\} <coproc>, \{#\}<opc1>, <Rt>, <Rt2>, <CRm>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1100</td>
<td>0b0001</td>
</tr>
</tbody>
</table>

if \( \text{PSTATE.EL} == \text{EL0} \) then
  UNDEFINED;
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
  if \( \text{EL2Enabled()} \&\& ![ELUsingAArch32(EL2) \&\& \text{HSTR.EL2.T12} == '1'] \) then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif \( \text{EL2Enabled()} \&\& ![ELUsingAArch32(EL2) \&\& \text{HSTR.T12} == '1'] \) then
    AArch32.TakeHypTrapException(0x03);
  elsif \( \text{ICC_SRE.SRE} == '0' \) then
    UNDEFINED;
  elsif \( \text{EL2Enabled()} \&\& ![ELUsingAArch32(EL2) \&\& \text{ICH_HCR_EL2.TC} == '1'] \) then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif \( \text{EL2Enabled()} \&\& ![ELUsingAArch32(EL2) \&\& \text{ICH.CR} == '1'] \) then
    AArch32.TakeHypTrapException(0x03);
  elsif \( \text{EL2Enabled()} \&\& ![ELUsingAArch32(EL2) \&\& \text{HCR_EL2.FMO} == '1'] \) then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif \( \text{EL2Enabled()} \&\& ![ELUsingAArch32(EL2) \&\& \text{HCR_EL2.IMO} == '1'] \) then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif \( \text{HaveEL(EL3) \&\& ![ELUsingAArch32(EL3) \&\& \text{SCR_EL3.<IRQ,FIQ>} == '11'] \) then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif \( \text{HaveEL(EL3) \&\& ![ELUsingAArch32(EL3) \&\& \text{SCR.<IRQ,FIQ>} == '11'] \) then
    AArch32.TakeMonitorTrapException();
else
  ICC_ASGI1R = R[t2]:R[t];
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
  if \( \text{ICC_HSRE.SRE} == '0' \) then
    UNDEFINED;
  elsif \( \text{HaveEL(EL3) \&\& ![ELUsingAArch32(EL3) \&\& \text{SCR_EL3.<IRQ,FIQ>} == '11'] \) then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif \( \text{HaveEL(EL3) \&\& ![ELUsingAArch32(EL3) \&\& \text{SCR.<IRQ,FIQ>} == '11'] \) then
    AArch32.TakeMonitorTrapException();
else
  ICC_ASGI1R = R[t2]:R[t];
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
  if \( \text{ICC_MSRE.SRE} == '0' \) then
    UNDEFINED;
else
  ICC_ASGI1R = R[t2]:R[t];
The ICC_BPR0 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

**Configuration**

AArch32 System register ICC_BPR0 bits [31:0] are architecturally mapped to AArch64 System register ICC_BPR0_EL1[31:0].

RW fields in this register reset to architecturally unknown values.

**Attributes**

ICC_BPR0 is a 32-bit register.

**Field descriptions**

The ICC_BPR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>BinaryPoint</td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>[7:1]</td>
</tr>
<tr>
<td>7</td>
<td>[7:2]</td>
</tr>
<tr>
<td>6</td>
<td>[7:3]</td>
</tr>
<tr>
<td>5</td>
<td>[7:4]</td>
</tr>
<tr>
<td>4</td>
<td>[7:5]</td>
</tr>
<tr>
<td>3</td>
<td>[7:6]</td>
</tr>
<tr>
<td>2</td>
<td>[7]</td>
</tr>
<tr>
<td>1</td>
<td>[6:0]</td>
</tr>
</tbody>
</table>

This field resets to an architecturally unknown value.

**Accessing the ICC_BPR0**

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is implementation defined, and reported by ICC_CTLR.PRIbits and ICC_MCTLR.PRIbits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is set to the minimum supported value.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
    return ICV_BPR0;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_BPR0;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    return ICV_BPR0;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_BPR0;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_BPR0;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICC_BPR0;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
  ICC_BPR0 = R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
  ICC_BPR0 = R[t];
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
  AArch32.TakeMonitorTrapException();
else
  ICC_BPR0 = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
  AArch32.TakeMonitorTrapException();
else
  ICC_BPR0 = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  ICC_BPR0 = R[t];
The ICC_BPR1 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

**Configuration**

AArch32 System register ICC_BPR1 bits [31:0] (S) are architecturally mapped to AArch64 System register ICC_BPR1_EL1[31:0] (S).

AArch32 System register ICC_BPR1 bits [31:0] (NS) are architecturally mapped to AArch64 System register ICC_BPR1_EL1[31:0] (NS).

In GIC implementations supporting two Security states, this register is Banked.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally unknown values.

**Attributes**

ICC_BPR1 is a 32-bit register.

**Field descriptions**

The ICC_BPR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 | BinaryPoint |

**Bits [31:3]**

Reserved, RES0.

**BinaryPoint, bits [2:0]**

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. For more information about priorities, see Priority grouping.

Writing 0 to this field will set this field to its reset value.

If EL3 is implemented and ICC_MCTLR_CBPR_EL1S is 1:

- Accesses to this register at EL3 not in Monitor mode access the state of ICC_BPR0.
- When SCR_EL3.EEL2 is 1 and HCR_EL2.IMO is 1, Secure accesses to this register at EL1 access the state of ICC_BPR1.
- Otherwise, Secure accesses to this register at EL1 access the state of ICC_BPR0.

If EL3 is implemented and ICC_MCTLR_CBPR_EL1NS is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR.IMO and SCR.IRQ:
If EL3 is not implemented and ICC_CTLR.CBPR is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR.IMO:

<table>
<thead>
<tr>
<th>HCR.IMO</th>
<th>SCR_IRQ</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b0</td>
<td>Non-secure EL1 and EL2 reads return ( \text{ICC_BPR0} + 1 ) saturated to 0b111. Non-secure EL1 and EL2 writes are ignored.</td>
</tr>
<tr>
<td>0b0</td>
<td>0b1</td>
<td>Non-secure EL1 and EL2 accesses trap to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ( \text{ICC_BPR0} + 1 ) saturated to 0b111. Non-secure EL2 writes ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 accesses trap to EL3.</td>
</tr>
</tbody>
</table>

This field resets to an IMPLEMENTATION DEFINED non-zero value.

**Accessing the ICC_BPR1**

When the PE resets into an Exception level that is using AArch32, the reset value is equal to:

- For the Secure copy of the register, the minimum value of ICC_BPR0 plus one.
- For the Non-secure copy of the register, the minimum value of ICC_BPR0.

Where the minimum value of ICC_BPR0 is IMPLEMENTATION DEFINED.

If EL3 is not implemented:

- If the PE is Secure this reset value is (minimum value of ICC_BPR0 plus one).
- If the PE is Non-secure this reset value is (minimum value of ICC_BPR0).

An attempt to program the binary point field to a value less than the reset value sets the field to the reset value.

Accesses to this register use the following encodings:

\[ \text{MRC}\{<c>\}{<q>} \ <\text{coproc}> , \{#\}<\text{opc1}> , \ <\text{Rt}> , \ <\text{CRn}> , \ <\text{CRm}> , \{#\}<\text{opc2}> \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.IMO == '1' then
        return ICV_BPR1;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_BPR1;
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_MSRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_MSRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_MSRE.SRE == '0' then
        UNDEFINED;
elsif ICC_MSRE.SRE == '0' then
    if SCR.NS == '0' then
        return ICC_BPR1_S;
    else
        return ICC_BPR1_NS;
else
    return ICC_BPR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_BPR1_S;
        else
            return ICC_BPR1_NS;
else
    return ICC_BPR1;
else
    if PSTATE.EL == EL0 then
        UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif ICC_SRE.SRE == '0' then
            UNDEFINED;
        elsif ICC_SRE.SRE == '0' then
            UNDEFINED;
        elsif ICC_SRE.SRE == '0' then
            UNDEFINED;
        elsif ICC_SRE.SRE == '0' then
            UNDEFINED;
elsif ICC_SRE.SRE == '0' then
    if SCR.NS == '0' then
        return ICC_BPR1_S;
    else
        return ICC_BPR1_NS;
else
    return ICC_BPR1;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_MSRE.SRE == '0' then
        UNDEFINED;
    elseif ICC_MSRE.SRE == '0' then
        UNDEFINED;
    elsif ICC_MSRE.SRE == '0' then
        UNDEFINED;
elsif ICC_MSRE.SRE == '0' then
    if SCR.NS == '0' then
        return ICC_BPR1_S;
    else
        return ICC_BPR1_NS;
else
    return ICC_BPR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_BPR1_S;
        else
            return ICC_BPR1_NS;

MCR{<c>{<q}} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        if SCR.NS == '0' then
            ICC_BPR1_S = R[t];
        else
            ICC_BPR1_NS = R[t];
        else
            ICC_BPR1 = R[t];
        endif
    elsif PSTATE.EL == EL2 then
        if ICC_HSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_BPR1_S = R[t];
            else
                ICC_BPR1_NS = R[t];
            endif
        endif
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_BPR1_S = R[t];
            else
                ICC_BPR1_NS = R[t];
            endif
        endif
    endif
endif

**ICC_CTLR, Interrupt Controller Control Register**

The ICC_CTLR characteristics are:

**Purpose**

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

**Configuration**

AArch32 System register ICC_CTLR bits [31:0] (S) are architecturally mapped to AArch64 System register ICC_CTLR_EL1[31:0] (S).

AArch32 System register ICC_CTLR bits [31:0] (NS) are architecturally mapped to AArch64 System register ICC_CTLR_EL1[31:0] (NS).

**Attributes**

ICC_CTLR is a 32-bit register.

**Field descriptions**

The ICC_CTLR bit assignments are:

| Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit | Bit |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
| RES0| ExtRange | RSS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | PMHE | RES0 | EOImode | CBPR |

**Bits [31:20]**

Reserved, RES0.

**ExtRange, bit [19]**

Extended INTID range (read-only).

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>CPU interface does not support INTIDs in the range 1024..8191. Behaviour is <strong>UNPREDICTABLE</strong> if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.</td>
</tr>
<tr>
<td>0b1</td>
<td>CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation.</td>
</tr>
</tbody>
</table>

**Note**

Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.

If EL3 is implemented, ICC_CTLR_EL1.ExtRange is an alias of ICC_CTLR_EL3.ExtRange.

**RSS, bit [18]**

Range Selector Support. Possible values are:
RSS

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 15 are supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 255 are supported.</td>
</tr>
</tbody>
</table>

This bit is read-only.

**Bits [17:16]**

Reserved, RES0.

**A3V, bit [15]**

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and using AArch32, this bit is an alias of ICC_MCTLR.A3V.

If EL3 is implemented and using AArch64, this bit is an alias of ICC_CTLR_EL3.A3V.

**SEIS, bit [14]**

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports local generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic does not support local generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports local generation of SEIs.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and using AArch32, this bit is an alias of ICC_MCTLR.SEIS.

If EL3 is implemented and using AArch64, this bit is an alias of ICC_CTLR_EL3.SEIS.

**IDbits, bits [13:11]**

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If EL3 is implemented and using AArch32, this field is an alias of ICC_MCTLR.IDbits.

If EL3 is implemented and using AArch64, this field is an alias of ICC_CTLR_EL3.IDbits.

**PRIbits, bits [10:8]**

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

**Note**
This field always returns the number of priority bits implemented, regardless of the Security state of the access or the value of \texttt{GICD\_CTRL\_DS}.

The division between group priority and subpriority is defined in the binary point registers \texttt{ICC\_BPR0} and \texttt{ICC\_BPR1}.

If EL3 is implemented and using AArch32, physical accesses return the value from \texttt{ICC\_MCTLR\_PRI} bits.

If EL3 is implemented and using AArch64, physical accesses return the value from \texttt{ICC\_CTRL\_EL3\_PRI} bits.

If EL3 is not implemented, physical accesses return the value from this field.

\textbf{Bit [7]}

Reserved, \texttt{RES0}.

\textbf{PMHE, bit [6]}

Priority Mask Hint Enable. Controls whether the priority mask register is used as a hint for interrupt distribution:

<table>
<thead>
<tr>
<th>PMHE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables use of \texttt{ICC_PMR} as a hint for interrupt distribution.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables use of \texttt{ICC_PMR} as a hint for interrupt distribution.</td>
</tr>
</tbody>
</table>

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of \texttt{ICC\_MCTLR\_PMHE}.
- If EL3 is using AArch64, this bit is an alias of \texttt{ICC\_CTRL\_EL3\_PMHE}.
- If \texttt{GICD\_CTRL\_DS} == 0, this bit is read-only.
- If \texttt{GICD\_CTRL\_DS} == 1, this bit is read/write.

If EL3 is not implemented, it is \texttt{IMPLEMENTATION DEFINED} whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

\textbf{Bits [5:2]}

Reserved, \texttt{RES0}.

\textbf{EOImode, bit [1]}

EOI mode for the current Security state. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>\texttt{ICC_EOIR0} and \texttt{ICC_EOIR1} provide both priority drop and interrupt deactivation functionality. Accesses to \texttt{ICC_DIR} are \texttt{UNPREDICTABLE}.</td>
</tr>
<tr>
<td>0b1</td>
<td>\texttt{ICC_EOIR0} and \texttt{ICC_EOIR1} provide priority drop functionality only. \texttt{ICC_DIR} provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of \texttt{ICC\_MCTLR\_EOImode\_EL1\{S, NS\}} where S or NS corresponds to the current Security state.
- If EL3 is using AArch64, this bit is an alias of \texttt{ICC\_CTRL\_EL3\_EOImode\_EL1\{S, NS\}} where S or NS corresponds to the current Security state.

If EL3 is not implemented, it is \texttt{IMPLEMENTATION DEFINED} whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.
**CBPR, bit [0]**

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupts:

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_BPR0 determines the preemption group for Group 0 interrupts only.</td>
</tr>
<tr>
<td></td>
<td>ICC_BPR1 determines the preemption group for Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_BPR0 determines the preemption group for both Group 0 and Group 1 interrupts.</td>
</tr>
</tbody>
</table>

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of ICC_MCTLR.CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If EL3 is using AArch64, this bit is an alias of ICC_CTLR_EL3.CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If GICD_CTLR.DS == 0, this bit is read-only.
- If GICD_CTLR.DS == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

**Accessing the ICC_CTLR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_CTLR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        if SCR.NS == '0' then
            return ICC_CTLR_S;
        else
            return ICC_CTLR_NS;
        else
            return ICC_CTLR;
    elsif PSTATE.EL == EL2 then
        if ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_CTLR_NS;
        else
            return ICC_CTLR;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                return ICC_CTLR_S;
            else
                return ICC_CTLR_NS;
        
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICV_CTLR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICV_CTLR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    ICV_CTLR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    ICV_CTLR = R[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    if SCR.NS == '0' then
      ICC_CTLR_S = R[t];
    else
      ICC_CTLR_NS = R[t];
    else
      ICC_CTLR = R[t];
  else
    ICC_CTLR = R[t];
else
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    ICC_CTLR_NS = R[t];
  else
    ICC_CTLR = R[t];
else
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    if SCR.NS == '0' then
      ICC_CTLR_S = R[t];
    else
      ICC_CTLR_NS = R[t];
The ICC_DIR characteristics are:

**Purpose**

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

**Configuration**

AArch32 System register ICC_DIR performs the same function as AArch64 System register ICC_DIR_EL1.

**Attributes**

ICC_DIR is a 32-bit register.

**Field descriptions**

The ICC_DIR bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_DIR**

There are two cases when writing to ICC_DIR_EL1 that were UNPREDICTABLE for a corresponding GICv2 write to GICC_DIR:

- When EOImode == 0. GICv3 implementations must ignore such writes. In systems supporting system error generation, an implementation might generate an SEI.
- When EOImode == 1 but no EOI has been issued. The interrupt will be de-activated by the Distributor, however the active priority in the CPU interface for the interrupt will remain set (because no EOI was issued).

Accesses to this register use the following encodings:

\[\text{MCR}\{<c>\}{<q>} \{<coproc>, \{<opc1>, <Rt>, <CRn>, <CRm>{, \{<opc2>}\}\}\} \]

\[\begin{array}{|c|c|c|c|c|}
\hline
\text{coproc} & \text{opc1} & \text{CRn} & \text{CRm} & \text{opc2} \\
\hline
0b1111 & 0b000 & 0b1100 & 0b1011 & 0b001 \\
\hline
\end{array}\]
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TDIR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TDIR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICV_DIR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICV_DIR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    ICV_DIR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    ICV_DIR = R[t];
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
  AArch32.TakeMonitorTrapException();
else
  ICC_DIR = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
else
  ICC_DIR = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICC_DIR = R[t];
ICC_EOIR0, Interrupt Controller End Of Interrupt Register 0

The ICC_EOIR0 characteristics are:

**Purpose**

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 0 interrupt.

**Configuration**

AArch32 System register ICC_EOIR0 performs the same function as AArch64 System register ICC_EOIR0_EL1.

**Attributes**

ICC_EOIR0 is a 32-bit register.

**Field descriptions**

The ICC_EOIR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  | RES0 | INTID |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID from the corresponding ICC_IAR0 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR to deactivate the interrupt.

The appropriate EOImode bit varies as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR EOImode.
- If EL3 is implemented and the software is executing in Monitor mode, the appropriate bit is ICC_MCTLR EOImode_EL3.
- If EL3 is implemented and the software is not executing in Monitor mode, the bit depends on the current Security state:
  - If the software is executing in Secure state, the bit is ICC_CTLR EOImode in the Secure instance of ICC_CTLR. This is an alias of ICC_MCTLR EOImode_EL1S.
  - If the software is executing in Non-secure state, the bit is ICC_CTLR EOImode in the Non-secure instance of ICC_CTLR. This is an alias of ICC_MCTLR EOImode_EL1NS.
Accessing the ICC_EOIR0

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR0, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. See Special INTIDs, for more information.

Accesses to this register use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICC_HCR_EL2.FMO == '1' then
    ICC_EOIR0 = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICC_EOIR0 = R[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICC_EOIR0 = R[t];
ICC_EOIR1, Interrupt Controller End Of Interrupt Register 1

The ICC_EOIR1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 1 interrupt.

Configuration

AArch32 System register ICC_EOIR1 performs the same function as AArch64 System register ICC_EOIR1_EL1.

Attributes

ICC_EOIR1 is a 32-bit register.

Field descriptions

The ICC_EOIR1 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>INTID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding ICC_IAR1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to ICC_DIR to deactivate the interrupt.

The appropriate EOImode bit varies as follows:

- If EL3 is not implemented, the appropriate bit is ICC_CTLR EOImode.
- If EL3 is implemented and the software is executing in Monitor mode, the appropriate bit is ICC_MCTLR.EOImode_EL3.
- If EL3 is implemented and the software is not executing in Monitor mode, the bit depends on the current Security state:
  - If the software is executing in Secure state, the bit is ICC_CTLR.EOImode in the Secure instance of ICC_CTLR. This is an alias of ICC_MCTLR.EOImode_EL1S.
  - If the software is executing in Non-secure state, the bit is ICC_CTLR.EOImode in the Non-secure instance of ICC_CTLR. This is an alias of ICC_MCTLR.EOImode_EL1NS.
Accessing the ICC_EOIR1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICC_IAR1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. See Special INTIDs, for more information.

Accesses to this register use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif ICC_SRE.SRE == '0' then
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif ICC_SRE.SRE == '0' then
  UNDEFINED;
elsif ICC_HSRE.SRE == '0' then
  UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SC1.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ICC_EOIR1 = R[t];
else
  ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  ICC_EOIR1 = R[t];

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbf63431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**ICC_HPPIR0, Interrupt Controller Highest Priority Pending Interrupt Register 0**

The ICC_HPPIR0 characteristics are:

**Purpose**

Indicates the highest priority pending Group 0 interrupt on the CPU interface.

**Configuration**

AArch32 System register ICC_HPPIR0 performs the same function as AArch64 System register ICC_HPPIR0_EL1.

**Attributes**

ICC_HPPIR0 is a 32-bit register.

**Field descriptions**

The ICC_HPPIR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. See **Special INTIDs**, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_HPPIR0**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}, \text{coproc}, \{#\text{opc1}\}, <Rt>, <CRn>, <CRm>, \{#\text{opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b010</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_HPPIR0;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_HPPIR0;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return ICC_HPPIR0;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICC_HPPIR0;
The ICC_HPPIR1 characteristics are:

**Purpose**
Indicates the highest priority pending Group 1 interrupt on the CPU interface.

**Configuration**
AArch32 System register ICC_HPPIR1 performs the same function as AArch64 System register ICC_HPPIR1_EL1.

**Attributes**
ICC_HPPIR1 is a 32-bit register.

**Field descriptions**
The ICC_HPPIR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit位置</th>
<th>功能 Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>RESERVED RES0</td>
</tr>
<tr>
<td>23-0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**
Reserved, RES0.

**INTID, bits [23:0]**
The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_HPPIR1**
Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    if PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
      AArch32.TakeMonitorTrapException();
    else
      return ICC_HPPIR1;
  else
    if ICC_HSRE.SRE == '0' then
      UNDEFINED;
    elsif ICC_MSRE.SRE == '0' then
      if ICC_MSRE.SRE == '0' then
        UNDEFINED;
      else
        return ICC_HPPIR1;
      endif
    else
      return ICC_HPPIR1;
    endif
  endif
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_HPPIR1;
  endif
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICC_HPPIR1;
ICC_HSRE, Interrupt Controller Hyp System Register Enable register

The ICC_HSRE characteristics are:

**Purpose**

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

**Configuration**

AArch32 System register ICC_HSRE bits [31:0] are architecturally mapped to AArch64 System register ICC_SRE_EL2[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_HSRE is a 32-bit register.

**Field descriptions**

The ICC_HSRE bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Enable</td>
<td>DIB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**Enable, bit [3]**

Enable. Enables lower Exception level access to ICC_SRE.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 accesses to ICC_SRE trap to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to ICC_SRE do not trap to EL2.</td>
</tr>
</tbody>
</table>

If ICC_HSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_HSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

This field resets to an architecturally UNKNOWN value.

**DIB, bit [2]**

Disable IRQ bypass.

<table>
<thead>
<tr>
<th>DIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and GICD_CTLR.DS is 0, this field is a read-only alias of ICC_MSRE.DIB.
If EL3 is implemented and `GICD_CTLR.DS` is 1, this field is a read-write alias of `ICC_MSRE.DIB`. In systems that do not support IRQ bypass, this bit is RAO/WI.

This field resets to 0.

**DFB, bit [1]**

Disable FIQ bypass.

<table>
<thead>
<tr>
<th>DFB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and `GICD_CTLR.DS` is 0, this field is a read-only alias of `ICC_MSRE.DFB`.

In systems that do not support FIQ bypass, this bit is RAO/WI.

This field resets to 0.

**SRE, bit [0]**

System Register Enable.

<table>
<thead>
<tr>
<th>SRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The memory-mapped interface must be used. Accesses at EL2 or below to</td>
</tr>
<tr>
<td></td>
<td>any ICH_* System register, or any EL1 or EL2 ICC_* register other than</td>
</tr>
<tr>
<td></td>
<td><code>ICC_SRE</code> or <code>ICC_HSRE</code>, are UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>The System register interface to the ICH_* registers and the EL1 and</td>
</tr>
<tr>
<td></td>
<td>EL2 ICC_* registers is enabled for EL2.</td>
</tr>
</tbody>
</table>

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and using AArch64:

- When `ICC_SRE_EL3.SRE==0` this bit is RAZ/WI.

If EL3 is implemented using AArch32:

- When `ICC_MSRE.SRE==0` this bit is RAZ/WI.

This field resets to 0.

**Accessing the ICC_HSRE**

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while `ICC_HSRE.SRE==0`, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elseif ICC_MSRE.Enable == '0' then
        UNDEFINED;
    else
        return ICC_HSRE;
    end
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        return ICC_HSRE;
    end
MCR{<c>}{<q>}{<coproc>}, {#}<opc1>, <Rt>, <CRn>, <CRm}{, {#}<opc2>}

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
</tr>
<tr>
<td>0b101</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elseif ICC_MSRE.Enable == '0' then
        UNDEFINED;
    else
        ICC_HSRE = R[t];
    end
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        ICC_HSRE = R[t];
    end
The ICC_IAR0 characteristics are:

**Purpose**

The PE reads this register to obtain the INTID of the signaled Group 0 interrupt. This read acts as an acknowledge for the interrupt.

**Configuration**

AArch32 System register ICC_IAR0 performs the same function as AArch64 System register ICC_IAR0_EL1.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

**Attributes**

ICC_IAR0 is a 32-bit register.

**Field descriptions**

The ICC_IAR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23-0</td>
<td>INTID</td>
<td>The INTID of the signaled interrupt.</td>
</tr>
</tbody>
</table>

**INTID, bits [23:0]**

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_IAR0**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_IAR0;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_IAR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_IAR0;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_IAR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_IAR0;
ICC_IAR1, Interrupt Controller Interrupt Acknowledge Register 1

The ICC_IAR1 characteristics are:

**Purpose**

The PE reads this register to obtain the INTID of the signaled Group 1 interrupt. This read acts as an acknowledge for the interrupt.

**Configuration**

AArch32 System register ICC_IAR1 performs the same function as AArch64 System register ICC_IAR1_EL1.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.I,F == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

**Attributes**

ICC_IAR1 is a 32-bit register.

**Field descriptions**

The ICC_IAR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICC_CTLR.IDbits and ICC_MCTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICC_IAR1**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IAR1;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_IAR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_IAR1;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_IAR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_IAR1;
The ICC_IGRPEN0 characteristics are:

**Purpose**

Controls whether Group 0 interrupts are enabled or not.

**Configuration**

AArch32 System register ICC_IGRPEN0 bits [31:0] are architecturally mapped to AArch64 System register ICC_IGRPEN0_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_IGRPEN0 is a 32-bit register.

**Field descriptions**

The ICC_IGRPEN0 bit assignments are:

```
<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td>Enable</td>
<td></td>
</tr>
</tbody>
</table>
```

**Bits [31:1]**

Reserved, RES0.

**Enable, bit [0]**

Enables Group 0 interrupts.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

Virtual accesses to this register update ICH_VMCR.VENG0.

This field resets to 0.

**Accessing the ICC_IGRPEN0**

The lowest Exception level at which this register can be accessed is governed by the Exception level to which FIQ is routed. This routing depends on SCR.FIQ, SCR.NS and HCR.FMO.

If an interrupt is pending within the CPU interface when Enable becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_IGRPEN0;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    return ICV_IGRPEN0;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_IGRPEN0;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_IGRPEN0;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICC_IGRPEN0;

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR.EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR.EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICC_IGRPEN0 = R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICC_IGRPEN0 = R[t];
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
else
    ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
else
    ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
else
    ICC_IGRPEN0 = R[t];
ICC_IGRPEN1, Interrupt Controller Interrupt Group 1 Enable register

The ICC_IGRPEN1 characteristics are:

**Purpose**

Controls whether Group 1 interrupts are enabled for the current Security state.

**Configuration**

AArch32 System register ICC_IGRPEN1 bits [31:0] (S) are architecturally mapped to AArch64 System register ICC_IGRPEN1_EL1[31:0] (S).

AArch32 System register ICC_IGRPEN1 bits [31:0] (NS) are architecturally mapped to AArch64 System register ICC_IGRPEN1_EL1[31:0] (NS).

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICC_IGRPEN1 is a 32-bit register.

**Field descriptions**

The ICC_IGRPEN1 bit assignments are:

|   | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|   | RES0 | Enable |

**Bits [31:1]**

Reserved, RES0.

**Enable, bit [0]**

Enables Group 1 interrupts for the current Security state.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 interrupts are disabled for the current Security state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 interrupts are enabled for the current Security state.</td>
</tr>
</tbody>
</table>

Virtual accesses to this register update **ICH_VMCR.VENG1**.

If EL3 is present:

- This bit is a read/write alias of **ICC_MGRPEN1_EnableGrp1{S, NS}** as appropriate if EL3 is using AArch32, or **ICC_IGRPEN1_EL3_EnableGrp1{S, NS}** as appropriate if EL3 is using AArch64.
- When this register is accessed at EL3, the copy of this register appropriate to the current setting of SCR.NS is accessed.

This field resets to 0.
Accessing the ICC_IGRPEN1

The lowest Exception level at which this register can be accessed is governed by the Exception level to which IRQ is routed. This routing depends on SCR.IRQ, SCR.NS and HCR.IMO.

If an interrupt is pending within the CPU interface when Enable becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IGRPEN1;
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_IGRPEN1;
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IGRPEN1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        if SCR.NS == '0' then
            return ICC_IGRPEN1_S;
        else
            return ICC_IGRPEN1_NS;
        end else
            return ICC_IGRPEN1;
        elsif PSTATE.EL == EL2 then
            if ICC_HSRE.SRE == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) then
                if SCR.NS == '0' then
                    return ICC_IGRPEN1_S;
                else
                    return ICC_IGRPEN1_NS;
                end else
                    return ICC_IGRPEN1;
                elsif PSTATE.EL == EL3 then
                    if ICC_MSRE.SRE == '0' then
                        UNDEFINED;
                    else
                        if SCR.NS == '0' then
                            return ICC_IGRPEN1_S;
                        else
                            return ICC_IGRPEN1_NS;
                        end else
                            return ICC_IGRPEN1;
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_EL2.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN1_NS = R[t];
else
    ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN1_NS = R[t];
else
    ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    elseif SCR_NS == '0' then
        ICC_IGRPEN1_S = R[t];
    else
        ICC_IGRPEN1_NS = R[t];
else
    ICC_IGRPEN1 = R[t];
The ICC_MCTLR characteristics are:

**Purpose**

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

**Configuration**

AArch32 System register ICC_MCTLR bits [31:0] can be mapped to AArch64 System register ICC_CTLR_EL3[31:0], but this is not architecturally mandated.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICC_MCTLR is a 32-bit register.

**Field descriptions**

The ICC_MCTLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ExtRange | RSSnDS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | PMH | RM | E0Imode_EL1NS | E0Imode_EL1S | CBPR_EL1NS | CBPR_EL1S |

**Bits [31:20]**

Reserved, RES0.

**ExtRange, bit [19]**

Extended INTID range (read-only).

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>CPU interface does not support INTIDs in the range 1024..8191. Behavior is <strong>UNPREDICTABLE</strong> if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.</td>
</tr>
</tbody>
</table>

**Note**

Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.

| 0b1      | CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation. |

**RSS, bit [18]**

Range Selector Support. Possible values are:
RSS

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 15 are supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 255 are supported.</td>
</tr>
</tbody>
</table>

This bit is read-only.

**nDS, bit [17]**

Disable Security not supported. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>nDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic supports disabling of security.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic does not support disabling of security, and requires that security is not disabled.</td>
</tr>
</tbody>
</table>

**Bit [16]**

Reserved, RES0.

**A3V, bit [15]**

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic does not support non-zero values of the Aff3 field in SGI generation System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports non-zero values of the Aff3 field in SGI generation System registers.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTLR.AV3 is an alias of ICC_MCTLR.A3V

**SEIS, bit [14]**

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic does not support generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic supports generation of SEIs.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTLR.SEIS is an alias of ICC_MCTLR.SEIS

**IDbits, bits [13:11]**

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

If EL3 is present, ICC_CTLR.IDbits is an alias of ICC_MCTLR.IDbits

**PRIbits, bits [10:8]**

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).
This field always returns the number of priority bits implemented, regardless of the value of SCR.NS or the value of GICD_CTRLR.DS.

The division between group priority and subpriority is defined in the binary point registers ICC_BPR0 and ICC_BPR1. This field determines the minimum value of ICC_BPR0.

**Bit [7]**

Reserved, RES0.

**PMHE, bit [6]**

Priority Mask Hint Enable.

<table>
<thead>
<tr>
<th>PMHE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disables use of the priority mask register as a hint for interrupt distribution.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enables use of the priority mask register as a hint for interrupt distribution.</td>
</tr>
</tbody>
</table>

Software must write ICC_PMR to 0xFF before clearing this field to 0.

An implementation might choose to make this field RAO/WI.

If EL3 is present, ICC_CTRLR.PMHE is an alias of ICC_MCTLR.PMHE. This field resets to 0.

**RM, bit [5]**

SBZ.

The equivalent bit in AArch64 is the Routing Modifier bit. This feature is not supported when EL3 is using AArch32. This field resets to an architecturally UNKNOWN value.

**EOImode_EL1NS, bit [4]**

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode_EL1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_CTRLR(NS).EOImode is an alias of ICC_MCTLR.EOImode_EL1NS. This field resets to an architecturally UNKNOWN value.

**EOImode_EL1S, bit [3]**

EOI mode for interrupts handled at Secure EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt:
EOImode_EL1S, bit [0]

EOI mode for interrupts handled at EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode_EL1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

EOImode_EL3, bit [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

<table>
<thead>
<tr>
<th>EOImode_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

CBPR_EL1NS, bit [1]

Common Binary Point Register, EL1 Non-secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2:

<table>
<thead>
<tr>
<th>CBPR_EL1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Non-secure Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_BPR0 determines the preemption group for Group 0 interrupts and Non-secure Group 1 interrupts. Non-secure accesses to GICC_BPR and ICC_BPR1 access the state of ICC_BPR0.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_MCTRL(NS).CBPR is an alias of ICC_MCTRL.CBPR_EL1NS.

This field resets to an architecturally UNKNOWN value.

CBPR_EL1S, bit [0]

Common Binary Point Register, EL1 Secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupts in Secure non-Monitor modes:

<table>
<thead>
<tr>
<th>CBPR_EL1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Secure Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICC_BPR0 determines the preemption group for Group 0 interrupts and Secure Group 1 interrupts. Secure EL1 accesses, or EL3 accesses when not in Monitor mode, to ICC_BPR1 access the state of ICC_BPR0.</td>
</tr>
</tbody>
</table>

If EL3 is present, ICC_MCTRL(S).CBPR is an alias of ICC_MCTRL.CBPR_EL1S.

This field resets to an architecturally UNKNOWN value.
Accessing the ICC_MCTLR

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#\}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b110</td>
<td>0b110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#\}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b110</td>
<td>0b110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_MCTLR = R[t];
```
ICC_MGRPEN1, Interrupt Controller Monitor Interrupt Group 1 Enable register

The ICC_MGRPEN1 characteristics are:

**Purpose**

Controls whether Group 1 interrupts are enabled or not.

**Configuration**

AArch32 System register ICC_MGRPEN1 bits [31:0] can be mapped to AArch64 System register ICC_IGRPEN1_EL3[31:0], but this is not architecturally mandated.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_MGRPEN1 is a 32-bit register.

**Field descriptions**

The ICC_MGRPEN1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |    |    | EnableGrp1SEnableGrp1NS |

**Bits [31:2]**

Reserved, RES0.

**EnableGrp1S, bit [1]**

Enables Group 1 interrupts for the Secure state.

<table>
<thead>
<tr>
<th>EnableGrp1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

The Secure ICC_IGRPEN1.Enable bit is a read/write alias of the ICC_MGRPEN1.EnableGrp1S bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

This field resets to 0.

**EnableGrp1NS, bit [0]**

Enables Group 1 interrupts for the Non-secure state.

<table>
<thead>
<tr>
<th>EnableGrp1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

The Non-secure ICC_IGRPEN1.Enable bit is a read/write alias of the ICC_MGRPEN1.EnableGrp1NS bit.
If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

This field resets to 0.

## Accessing the ICC_MGRPEN1

If an interrupt is pending within the CPU interface when an Enable bit becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings:

\[
\text{MRC} \{<c>\} \{<q>\} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

If \( \text{PSTATE.EL} = \text{EL0} \) then

UNDEFINED;

elsif \( \text{PSTATE.EL} = \text{EL1} \) then

if \( \text{EL2Enabled()} \) && \( \text{!ELUsingAArch32(EL2)} \) && \( \text{HSTR_EL2.T12} = '1' \) then

\text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};

elsif \( \text{EL2Enabled()} \) && \( \text{ELUsingAArch32(EL2)} \) && \( \text{HSTR.T12} = '1' \) then

\text{AArch32.TakeHypTrapException(0x03)};

else

UNDEFINED;

elsif \( \text{PSTATE.EL} = \text{EL2} \) then

UNDEFINED;

elsif \( \text{PSTATE.EL} = \text{EL3} \) then

if \( \text{ICC_MSRE.SRE} = '0' \) then

UNDEFINED;

else

return ICC_MGRPEN1;

\[
\text{MCR} \{<c>\} \{<q>\} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

If \( \text{PSTATE.EL} = \text{EL0} \) then

UNDEFINED;

elsif \( \text{PSTATE.EL} = \text{EL1} \) then

if \( \text{EL2Enabled()} \) && \( \text{!ELUsingAArch32(EL2)} \) && \( \text{HSTR_EL2.T12} = '1' \) then

\text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};

elsif \( \text{EL2Enabled()} \) && \( \text{ELUsingAArch32(EL2)} \) && \( \text{HSTR.T12} = '1' \) then

\text{AArch32.TakeHypTrapException(0x03)};

else

UNDEFINED;

elsif \( \text{PSTATE.EL} = \text{EL2} \) then

UNDEFINED;

elsif \( \text{PSTATE.EL} = \text{EL3} \) then

if \( \text{ICC_MSRE.SRE} = '0' \) then

UNDEFINED;

else

\( \text{ICC_MGRPEN1} = \text{R[t]} \);

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The ICC_MSRE characteristics are:

**Purpose**

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

**Configuration**

AArch32 System register ICC_MSRE bits [31:0] can be mapped to AArch64 System register ICC_SRE_EL3[31:0], but this is not architecturally mandated.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICC_MSRE is a 32-bit register.

**Field descriptions**

The ICC_MSRE bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | Enable | DIB | DFB | SRE |

**Bits [31:4]**

Reserved, RES0.

**Enable, bit [3]**

Enable. Enables lower Exception level access to ICC_SRE and ICC_HSRE.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL1 accesses to Secure ICC_SRE trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE trap to EL3. Non-secure EL1 accesses to ICC_SRE trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_HSRE.Enable == 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL1 accesses to Secure ICC_SRE do not trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE do not trap to EL3. Non-secure EL1 accesses to ICC_SRE do not trap to EL3.</td>
</tr>
</tbody>
</table>

If ICC_MSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_MSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

This field resets to an architecturally **UNKNOWN** value.
**DIB, bit [2]**

Disable IRQ bypass.

<table>
<thead>
<tr>
<th>DIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQ bypass disabled.</td>
</tr>
</tbody>
</table>

In systems that do not support IRQ bypass, this bit is RAO/WI.

This field resets to 0.

**DFB, bit [1]**

Disable FIQ bypass.

<table>
<thead>
<tr>
<th>DFB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQ bypass disabled.</td>
</tr>
</tbody>
</table>

In systems that do not support FIQ bypass, this bit is RAO/WI.

This field resets to 0.

**SRE, bit [0]**

System Register Enable.

<table>
<thead>
<tr>
<th>SRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The memory-mapped interface must be used. Accesses at EL3 or below to any ICH_* System register, or any EL1, EL2, or EL3 ICC_* register other than ICC_SRE, ICC_HSRE, or ICC_MSRE, are UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3.</td>
</tr>
</tbody>
</table>

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

This field resets to 0.

**Accessing the ICC_MSRE**

This register is always System register accessible.

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_MSRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return ICC_MSRE;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
    UNDEFINED;
  else
    ICC_MSRE = R[t];
ICC_PMР, Interrupt Controller Interrupt Priority Mask Register

The ICC_PMР characteristics are:

**Purpose**

Provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

**Configuration**

AArch32 System register ICC_PMР bits [31:0] are architecturally mapped to AArch64 System register ICC_PMР_EL1[31:0].

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. See Observability of the effects of accesses to the GIC registers, for more information.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICC_PMР is a 32-bit register.

**Field descriptions**

The ICC_PMР bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

The possible priority field values are as follows:

<table>
<thead>
<tr>
<th>Implemented priority bits</th>
<th>Possible priority field values</th>
<th>Number of priority levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:0]</td>
<td>0x00-0xFF (0-255), all values</td>
<td>256</td>
</tr>
<tr>
<td>[7:1]</td>
<td>0x00-0xFE (0-254), even values only</td>
<td>128</td>
</tr>
<tr>
<td>[7:2]</td>
<td>0x00-0xFC (0-252), in steps of 4</td>
<td>64</td>
</tr>
<tr>
<td>[7:3]</td>
<td>0x00-0xF8 (0-248), in steps of 8</td>
<td>32</td>
</tr>
<tr>
<td>[7:4]</td>
<td>0x00-0xF0 (0-240), in steps of 16</td>
<td>16</td>
</tr>
</tbody>
</table>
Unimplemented priority bits are RAZ/WI.

This field resets to 0.

**Accessing the ICC_PMR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} \text{ <coproc>, \#<opc1>, <Rt>, <CRn>, <CRm>\{, \#<opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_PMR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_PMR;
    elsif PSTATE.EL == EL2 then
        if ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            AArch32.TakeMonitorTrapException();
        else
            return ICC_PMR;
        elsif PSTATE.EL == EL3 then
            if ICC_MSRE.SRE == '0' then
                UNDEFINED;
            else
                return ICC_PMR;
            \]

\[
\text{MCR}\{<c>\}\{<q>\} \text{ <coproc>, \#<opc1>, <Rt>, <CRn>, <CRm>\{, \#<opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
else
  ICC_PMR = R[t];
endif;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
else
  ICC_PMR = R[t];
endif;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICC_PMR = R[t];
endif;
**ICC_RPR, Interrupt Controller Running Priority Register**

The ICC_RPR characteristics are:

**Purpose**

Indicates the Running priority of the CPU interface.

**Configuration**

AArch32 System register ICC_RPR performs the same function as AArch64 System register ICC_RPR_EL1.

**Attributes**

ICC_RPR is a 32-bit register.

**Field descriptions**

The ICC_RPR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 | Priority |

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

---

**Note**

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

---

**Accessing the ICC_RPR**

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\{, \{#\}<\text{opc2}>}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_RPR;
  elsif EL2Enabled() && HCR_EL2.IMO == '1' then
    return ICV_RPR;
else
  return ICC_RPR;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
else
  return ICC_RPR;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  return ICC_RPR;
ICC_SGI0R, Interrupt Controller Software Generated Interrupt Group 0 Register

The ICC_SGI0R characteristics are:

**Purpose**

Generates Secure Group 0 SGIs.

**Configuration**

AArch32 System register ICC_SGI0R performs the same function as AArch64 System register ICC_SGI0R_EL1.

**Attributes**

ICC_SGI0R is a 64-bit register.

**Field descriptions**

The ICC_SGI0R bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47-44</td>
<td>RS</td>
</tr>
<tr>
<td>43-41</td>
<td>INTID</td>
</tr>
<tr>
<td>40-33</td>
<td>Aff1</td>
</tr>
<tr>
<td>32-24</td>
<td>TargetList</td>
</tr>
<tr>
<td>23-16</td>
<td>Aff2</td>
</tr>
<tr>
<td>15-8</td>
<td>RS</td>
</tr>
<tr>
<td>7-4</td>
<td>RES0</td>
</tr>
<tr>
<td>3-0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [63:56]**

Reserved, RES0.

**Aff3, bits [55:48]**

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**RS, bits [47:44]**

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTRLR_EL1.RSS==0, RS is RES0.

When ICC_CTRLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a constrained unpredictable choice of:

- The write is ignored.
- The RS field is treated as 0.

**Bits [43:41]**

Reserved, RES0.
**Interrupt Routing Mode.** Determines how the generated interrupts are distributed to PEs. Possible values are:

<table>
<thead>
<tr>
<th>IRM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.&lt;target list&gt;</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to all PEs in the system, excluding &quot;self&quot;.</td>
</tr>
</tbody>
</table>

**Aff2, bits [39:32]**

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**Bits [31:28]**

Reserved, RES0.

**INTID, bits [27:24]**

The INTID of the SGI.

**Aff1, bits [23:16]**

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**TargetList, bits [15:0]**

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

**Note**

This restricts a system to sending targeted SGIs to PEs with an affinity 0 number that is less than 16. If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

**Accessing the ICC_SGI0R**

This register allows software executing in a Secure state to generate Group 0 SGIs. It will also allow software executing in a Non-secure state to generate Group 0 SGIs, if permitted by the settings of **GICR_NSACR** in the Redistributor corresponding to the target PE.

When **GICD_CTLR**.DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the **GICR_NSACR** register associated with the target PE. For more information see Use of control registers for SGI forwarding.

**Note**
Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings:

\[
\text{MCRR}\{c\}\{q\} \begin{array}{c|c|c}
\text{coproc} & \text{CRm} & \text{opc1} \\
\hline
0b1111 & 0b1100 & 0b0010
\end{array}
\]

if \(\text{PSTATE.EL} == \text{EL0}\) then
UNDEFINED;
elsif \(\text{PSTATE.EL} == \text{EL1}\) then
  if \(\text{EL2Enabled()} \&\& \text{!ELUsingAArch32(EL2)} \&\& \text{HSTR_EL2.T12} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR.T12} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \(\text{ICC_SRE.SRE} == '0'\) then
    UNDEFINED;
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{ICH_HCR_EL2.TC} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{ICC_SRE.SRE} == '0'\) then
    UNDEFINED;
  elsif \(\text{EL2Enabled()} \&\& \text{!ELUsingAArch32(EL2)} \&\& \text{ICH_HCR.TC} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR_EL2.FMO} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR_EL2.IMO} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR.FMO} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR.IMO} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \(\text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR_EL3.<IRQ,FIQ>} == '11'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  elsif \(\text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR.<IRQ,FIQ>} == '11'\) then
    \text{AArch32.TakeMonitorTrapException();}
  else
    \text{ICC_SGI0R} = R[t2]:R[t];

elsif \(\text{PSTATE.EL} == \text{EL2}\) then
  if \(\text{ICC_HSRE.SRE} == '0'\) then
    UNDEFINED;
  elsif \(\text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR_EL3.<IRQ,FIQ>} == '11'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  elsif \(\text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR.<IRQ,FIQ>} == '11'\) then
    \text{AArch32.TakeMonitorTrapException();}
  else
    \text{ICC_SGI0R} = R[t2]:R[t];

elsif \(\text{PSTATE.EL} == \text{EL3}\) then
  if \(\text{ICC_MSRE.SRE} == '0'\) then
    UNDEFINED;
  else
    \text{ICC_SGI0R} = R[t2]:R[t];
ICC_SGI1R, Interrupt Controller Software Generated Interrupt Group 1 Register

The ICC_SGI1R characteristics are:

**Purpose**

Generates Group 1 SGIs for the current Security state.

**Configuration**

AArch32 System register ICC_SGI1R performs the same function as AArch64 System register ICC_SGI1R_EL1.

Under certain conditions a write to ICC_SGI1R can generate Group 0 interrupts, see Forwarding an SGI to a target PE.

**Attributes**

ICC_SGI1R is a 64-bit register.

**Field descriptions**

The ICC_SGI1R bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Aff3 | RES0 | Aff1 | RS | RES0 | IRM | Aff2 | TargetList |

**Bits [63:56]**

Reserved, RES0.

**Aff3, bits [55:48]**

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**RS, bits [47:44]**

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value ((RS * 16) + n).

When ICC_CTLR_EL1.RSS==0, RS is RES0.

When ICC_CTLR_EL1.RSS==1 and GICD_TYPER.RSS==0, writing this register with RS != 0 is a CONSTRAINED UNPREDICTABLE choice of:

- The write is ignored.
- The RS field is treated as 0.
**Bits [43:41]**

Reserved, RES0.

**IRM, bit [40]**

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

<table>
<thead>
<tr>
<th>IRM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.&lt;target list&gt;.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to all PEs in the system, excluding &quot;self&quot;.</td>
</tr>
</tbody>
</table>

**Aff2, bits [39:32]**

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**Bits [31:28]**

Reserved, RES0.

**INTID, bits [27:24]**

The INTID of the SGI.

**Aff1, bits [23:16]**

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

**TargetList, bits [15:0]**

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATIONDEFINED whether, in such cases, a Distributor can signal a system error.

**Note**

This restricts a system to sending targeted SGIs to PEs with an affinity 0 number that is less than 16. If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

**Accessing the ICC_SGI1R**

**Note**

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.
Accesses to this register use the following encodings:

\[
\text{MCRR}\{<c>\}{<q>} \text{ <coproc>}, \{#\}\text{<opc1>}, <\text{Rt}>, <\text{Rt2}>, <\text{CRM}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1100</td>
<td>0b0000</td>
</tr>
</tbody>
</table>

if \( \text{PSTATE.EL} == \text{EL0} \) then
  UNDEFINED;
elsif \( \text{PSTATE.EL} == \text{EL1} \) then
  if \( \text{EL2Enabled()} \land \text{!ELUsingAArch32(EL2)} \land \text{HSTR_EL2.T12} == '1' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \( \text{EL2Enabled()} \land \text{ELUsingAArch32(EL2)} \land \text{HSTR.T12} == '1' \) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \( \text{ICC_SRE.SRE} == '0' \) then
    UNDEFINED;
  elsif \( \text{EL2Enabled()} \land \text{!ELUsingAArch32(EL2)} \land \text{ICH_HCR_EL2.TC} == '1' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \( \text{EL2Enabled()} \land \text{ELUsingAArch32(EL2)} \land \text{ICH_HCR.TC} == '1' \) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \( \text{EL2Enabled()} \land \text{ELUsingAArch32(EL2)} \land \text{HCR_EL2.FMO} == '1' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \( \text{EL2Enabled()} \land \text{ELUsingAArch32(EL2)} \land \text{HCR_EL2.IMO} == '1' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \( \text{EL2Enabled()} \land \text{ELUsingAArch32(EL2)} \land \text{HCR.FMO} == '1' \) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \( \text{EL2Enabled()} \land \text{ELUsingAArch32(EL2)} \land \text{HCR.IMO} == '1' \) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \( \text{HaveEL(EL3)} \land \text{!ELUsingAArch32(EL3)} \land \text{SCR_EL3.<IRQ,FIQ>} == '11' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  elsif \( \text{HaveEL(EL3)} \land \text{ELUsingAArch32(EL3)} \land \text{SCR.<IRQ,FIQ>} == '11' \) then
    \text{AArch32.TakeMonitorTrapException();}
else
  \text{ICC_SGI1R} = \text{R[t2]:R[t]};
elsif \( \text{PSTATE.EL} == \text{EL2} \) then
  if \( \text{ICC_HSRE.SRE} == '0' \) then
    UNDEFINED;
  elsif \( \text{HaveEL(EL3)} \land \text{!ELUsingAArch32(EL3)} \land \text{SCR_EL3.<IRQ,FIQ>} == '11' \) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  elsif \( \text{HaveEL(EL3)} \land \text{ELUsingAArch32(EL3)} \land \text{SCR.<IRQ,FIQ>} == '11' \) then
    \text{AArch32.TakeMonitorTrapException();}
else
  \text{ICC_SGI1R} = \text{R[t2]:R[t]};
elsif \( \text{PSTATE.EL} == \text{EL3} \) then
  if \( \text{ICC_MSRE.SRE} == '0' \) then
    UNDEFINED;
  else
    \text{ICC_SGI1R} = \text{R[t2]:R[t]};
}
ICC_SRE, Interrupt Controller System Register Enable register

The ICC_SRE characteristics are:

**Purpose**

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL0 and EL1.

**Configuration**

AArch32 System register ICC_SRE bits [31:0] (S) are architecturally mapped to AArch64 System register ICC_SRE_EL1[31:0] (S).

AArch32 System register ICC_SRE bits [31:0] (NS) are architecturally mapped to AArch64 System register ICC_SRE_EL1[31:0] (NS).

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICC_SRE is a 32-bit register.

**Field descriptions**

The ICC_SRE bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>DIB</td>
<td>DFB</td>
<td>SRE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**DIB, bit [2]**

Disable IRQ bypass.

<table>
<thead>
<tr>
<th>DIB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and GICD_CTLR_DS == 0, this field is a read-only alias of ICC_MSRE.DIB.

If EL3 is implemented and GICD_CTLR_DS == 1, and EL2 is not implemented, this field is a read-write alias of ICC_MSRE.DIB.

If EL3 is not implemented or GICD_CTLR_DS == 1, and EL2 is implemented, this field is a read-only alias of ICC_HSRE.DIB.

In systems that do not support IRQ bypass, this field is RAO/WI.

This field resets to 0.
**DFB, bit [1]**

Disable FIQ bypass.

<table>
<thead>
<tr>
<th>DFB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQ bypass enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQ bypass disabled.</td>
</tr>
</tbody>
</table>

If EL3 is implemented and $GICD\_CTRL.DS == 0$, this field is a read-only alias of $ICC\_MSRE.DFB$. If EL3 is implemented and $GICD\_CTRL.DS == 1$, and EL2 is not implemented, this field is a read-write alias of $ICC\_MSRE.DFB$. If EL3 is not implemented or $GICD\_CTRL.DS == 1$, and EL2 is implemented, this field is a read-only alias of $ICC\_HSRE.DFB$. In systems that do not support FIQ bypass, this field is RAO/WI. This field resets to 0.

**SRE, bit [0]**

System Register Enable.

<table>
<thead>
<tr>
<th>SRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The memory-mapped interface must be used. Accesses at EL1 to any ICC_* System register other than ICC_SRE are UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>The System register interface for the current Security state is enabled.</td>
</tr>
</tbody>
</table>

If software changes this bit from 1 to 0 in the Secure instance of this register, the results are UNPREDICTABLE. If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI. If EL3 is implemented and using AArch64:

- When $ICC\_SRE\_EL3.SRE==0$ the Secure copy of this bit is RAZ/WI.
- When $ICC\_SRE\_EL3.SRE==0$ the Non-secure copy of this bit is RAZ/WI.

If EL3 is implemented and using AArch32:

- When $ICC\_MSRE.SRE==0$ the Secure copy of this bit is RAZ/WI.
- When $ICC\_MSRE.SRE==0$ the Non-secure copy of this bit is RAZ/WI.

If EL2 is implemented and using AArch64:

- When $ICC\_SRE\_EL2.SRE==0$ the Non-secure copy of this bit is RAZ/WI.

If EL2 is implemented and using AArch32:

- When $ICC\_HSRE.SRE==0$ the Non-secure copy of this bit is RAZ/WI.

This field resets to 0.

**Accessing the ICC_SRE**

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_SRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

Accesses to this register use the following encodings:
MRC{<c>}{<q>}, <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif ICC_MSRE.Enable == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) then
    if SCR_EL3.NS == '0' then
      ICC_SRE_NS = R[t];
    else
      ICC_SRE_NS = R[t];
  else
    ICC_SRE_NS = R[t];
  endif
elsif PSTATE.EL == EL3 then
  if SCR_EL3.NS == '0' then
    ICC_SRE_S = R[t];
  else
    ICC_SRE_NS = R[t];
  endif
endif
ICH_AP0R<n>, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n> characteristics are:

**Purpose**

Provides information about Group 0 active priorities for EL2.

**Configuration**

AArch32 System register ICH_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_AP0R<n>_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_AP0R<n> is a 32-bit register.

**Field descriptions**

The ICH_AP0R<n> bit assignments are:

<table>
<thead>
<tr>
<th>P&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no Group 0 interrupt active at the priority corresponding to that bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>There is a Group 0 interrupt active at the priority corresponding to that bit.</td>
</tr>
</tbody>
</table>

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP0R1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP0R1 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2 in the bits corresponding to 10:Priority[5:1].
The active state of preemption levels 192 - 254 are held in ICH_AP0R3 in the bits corresponding to 11:Priority[5:1].

**Note**

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n> and ICH_AP1R<n> might result in **UNPREDICTABLE** behavior of the interrupt prioritization system for virtual interrupts.

This field resets to 0.

### Accessing the ICH_AP0R<n>

ICH_AP0R1 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP0R2 and ICH_AP0R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are **UNDEFINED**.

**Note**

The number of bits of preemption is indicated by ICH_VTR.PREbits

Writing to the active priority registers in any order other than the following order will result in **UNPREDICTABLE** behavior:

- ICH_AP0R<n>
- ICH_AP1R<n>

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
  endif
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_AP0R[UInt(opc2<1:0>)];
  endif
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_AP0R[UInt(opc2<1:0>)];
  endif
endif

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICH_AP0R[UInt(opc2<1:0>)] = R[t];
        end elsif
    end if
end if
ICH_AP1R<n>, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n> characteristics are:

**Purpose**

Provides information about Group 1 active priorities for EL2.

**Configuration**

AArch32 System register ICH_AP1R<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_AP1R<n>_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_AP1R<n> is a 32-bit register.

**Field descriptions**

The ICH_AP1R<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>P&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0b0</td>
<td>There is no Group 1 interrupt active at the priority corresponding to that bit.</td>
</tr>
<tr>
<td>31</td>
<td>0b1</td>
<td>There is a Group 1 interrupt active at the priority corresponding to that bit.</td>
</tr>
</tbody>
</table>

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP1R1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP1R0 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP1R1 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP1R2 in the bits corresponding to 10:Priority[5:1].
• The active state of preemption levels 192 - 254 are held in ICH_AP1R3 in the bits corresponding to 11:Priority[5:1].

**Note**

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n> and ICH_AP1R<n> might result in **UNPREDICTABLE** behavior of the interrupt prioritization system for virtual interrupts.

This field resets to 0.

### Accessing the ICH_AP1R<n>

ICH_AP1R1 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP1R2 and ICH_AP1R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are **UNDEFINED**.

**Note**

The number of bits of preemption is indicated by ICH_VTR.PREbits

Writing to the active priority registers in any order other than the following order will result in **UNPREDICTABLE** behavior:

- ICH_AP0R<n>
- ICH_AP1R<n>

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\}{, \{#\}<\text{opc2}\} \\
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
  end if;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_AP1R[UInt(opc2<1:0>)];
  end if;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_AP1R[UInt(opc2<1:0>)];
  end if;
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICH_AP1R[UInt(opc2<1:0>)] = R[t];
    end
end
ICH_EISR, Interrupt Controller End of Interrupt Status Register

The ICH_EISR characteristics are:

**Purpose**

Indicates which List registers have outstanding EOI maintenance interrupts.

**Configuration**

AArch32 System register ICH_EISR bits [31:0] are architecturally mapped to AArch64 System register ICH_EISR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_EISR is a 32-bit register.

**Field descriptions**

The ICH_EISR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>Status&lt;0&gt;, bit [0]</td>
</tr>
<tr>
<td>29</td>
<td>Status&lt;1&gt;, bit [1]</td>
</tr>
<tr>
<td>28</td>
<td>Status&lt;2&gt;, bit [2]</td>
</tr>
<tr>
<td>27</td>
<td>Status&lt;3&gt;, bit [3]</td>
</tr>
<tr>
<td>26</td>
<td>Status&lt;4&gt;, bit [4]</td>
</tr>
<tr>
<td>25</td>
<td>Status&lt;5&gt;, bit [5]</td>
</tr>
<tr>
<td>24</td>
<td>Status&lt;6&gt;, bit [6]</td>
</tr>
<tr>
<td>23</td>
<td>Status&lt;7&gt;, bit [7]</td>
</tr>
<tr>
<td>22</td>
<td>Status&lt;8&gt;, bit [8]</td>
</tr>
<tr>
<td>21</td>
<td>Status&lt;9&gt;, bit [9]</td>
</tr>
<tr>
<td>20</td>
<td>Status&lt;10&gt;, bit [10]</td>
</tr>
<tr>
<td>19</td>
<td>Status&lt;11&gt;, bit [11]</td>
</tr>
<tr>
<td>18</td>
<td>Status&lt;12&gt;, bit [12]</td>
</tr>
<tr>
<td>17</td>
<td>Status&lt;13&gt;, bit [13]</td>
</tr>
<tr>
<td>16</td>
<td>Status&lt;14&gt;, bit [14]</td>
</tr>
<tr>
<td>15</td>
<td>Status&lt;15&gt;, bit [15]</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 0 to 15**

EOI maintenance interrupt status bit for List register <n>:

<table>
<thead>
<tr>
<th>Status&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>List register &lt;n&gt;, ICH_LR&lt;n&gt;, does not have an EOI maintenance interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>List register &lt;n&gt;, ICH_LR&lt;n&gt;, has an EOI maintenance interrupt that has not been handled.</td>
</tr>
</tbody>
</table>

For any ICH_LR<n>, the corresponding status bit is set to 1 if all of the following are true:

- ICH_LRC<n>.State is 0b00.
- ICH_LRC<n>.HW is 0.
- ICH_LRC<n>.EOI (bit [9]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

This field resets to 0.

**Accessing the ICH_EISR**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_EISR;
else
  PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
      UNDEFINED;
    else
      return ICH_EISR;
ICH_ELRSR, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR characteristics are:

**Purpose**

Indicates which List registers contain valid interrupts.

**Configuration**

AArch32 System register ICH_ELRSR bits [31:0] are architecturally mapped to AArch64 System register ICH_ELRSR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

**Attributes**

ICH_ELRSR is a 32-bit register.

**Field descriptions**

The ICH_ELRSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>RES0</td>
</tr>
<tr>
<td>16-0</td>
<td>Status&lt;n&gt;, bit [n], for n = 0 to 15</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 0 to 15**

Status bit for List register <n>, ICH_LR<n>:

<table>
<thead>
<tr>
<th>Status&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>List register ICH_LR&lt;n&gt;, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>List register ICH_LR&lt;n&gt; does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt.</td>
</tr>
</tbody>
</table>

For any List register <n>, the corresponding status bit is set to 1 if ICH_LRC<n>.State is 0b00 and either ICH_LRC<n>.HW is 1 or ICH_LRC<n>.EOI (bit [9]) is 0.

**Accessing the ICH_ELRSR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
      AArch32.TakeHypTrapException(0x03);
   else
      UNDEFINED;
   end if;
elsif PSTATE.EL == EL2 then
   if ICC_HSRE.SRE == '0' then
      UNDEFINED;
   else
      return ICH_ELRSR;
   end if;
elsif PSTATE.EL == EL3 then
   if ICC_MSRE.SRE == '0' then
      UNDEFINED;
   else
      return ICH_ELRSR;
   end if;
ICH_HCR, Interrupt Controller Hyp Control Register

The ICH_HCR characteristics are:

**Purpose**

Controls the environment for VMs.

**Configuration**

AArch32 System register ICH_HCR bits [31:0] are architecturally mapped to AArch64 System register ICH_HCR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_HCR is a 32-bit register.

**Field descriptions**

The ICH_HCR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
| EOIcount | RES0 | TDIR | TSE | TALL1 | TALL0 | TCRES0 | VGrp1DIE | VGrp1EIE | VGrp0DIE | VGrp0EIE | NPIE | LRENPIE | UIE | En |

**EOIcount, bits [31:27]**

This field is incremented whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is CONSTRAINED UNPREDICTABLE whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers (ICH_AP0R<n>/ICH_AP1R<n>) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

This field resets to 0.

**Bits [26:15]**

Reserved, RES0.

**TDIR, bit [14]**

Trap Non-secure EL1 writes to ICC_DIR and ICV_DIR.
ICH_HCR, Interrupt Controller Hyp Control Register

<table>
<thead>
<tr>
<th>TDIR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 writes of ICC_DIR and ICV_DIR are not trapped to EL2, unless trapped by other mechanisms.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 writes of ICV_DIR are trapped to EL2. It is IMPLEMENTATION DEFINED whether Non-secure writes of ICC_DIR are trapped. Not trapping ICC_DIR writes is DEPRECATED.</td>
</tr>
</tbody>
</table>

Support for this bit is OPTIONAL, with support indicated by ICH_VTR.

If the implementation does not support this trap, this bit is RES0.

Arm deprecates not including this trap bit.

This field resets to 0.

TSEI, bit [13]

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at Non-secure EL1.

<table>
<thead>
<tr>
<th>TSEI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Locally generated SEIs do not cause a trap to EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Locally generated SEIs trap to EL2.</td>
</tr>
</tbody>
</table>

If ICH_VTR.SEIS is 0, this bit is RES0.

This field resets to 0.

TALL1, bit [12]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

<table>
<thead>
<tr>
<th>TALL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2.</td>
</tr>
</tbody>
</table>

This field resets to 0.

TALL0, bit [11]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

<table>
<thead>
<tr>
<th>TALL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2.</td>
</tr>
</tbody>
</table>

This field resets to 0.

TC, bit [10]

Trap all Non-secure EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.

<table>
<thead>
<tr>
<th>TC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 accesses to common registers proceed as normal.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 accesses to common registers trap to EL2.</td>
</tr>
</tbody>
</table>

This affects accesses to ICC_SGI0R, ICC_SGI1R, ICC_ASGI1R, ICC_CTLR, ICC_DIR, ICC_PMR, ICC_RPR, ICV_CTLR, ICV_DIR, ICV_PMR, and ICV_RPR.

This field resets to 0.
Reserved, RES0.

**VGrp1DIE, bit [7]**

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

<table>
<thead>
<tr>
<th>VGrp1DIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR.VENG1 is 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**VGrp1EIE, bit [6]**

VM Group 1 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

<table>
<thead>
<tr>
<th>VGrp1EIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR.VENG1 is 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**VGrp0DIE, bit [5]**

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

<table>
<thead>
<tr>
<th>VGrp0DIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR.VENG0 is 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**VGrp0EIE, bit [4]**

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

<table>
<thead>
<tr>
<th>VGrp0EIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when ICH_VMCR.VENG0 is 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**NPIE, bit [3]**

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

<table>
<thead>
<tr>
<th>NPIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.</td>
</tr>
</tbody>
</table>

This field resets to 0.
**LRENPIE, bit [2]**

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

<table>
<thead>
<tr>
<th>LRENPIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt is asserted while the EOIcount field is not 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**UIE, bit [1]**

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

<table>
<thead>
<tr>
<th>UIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**En, bit [0]**

Enable. Global enable bit for the virtual CPU interface:

<table>
<thead>
<tr>
<th>En</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual CPU interface operation disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual CPU interface operation enabled.</td>
</tr>
</tbody>
</table>

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of ICV_IAR0, ICV_IAR1, GICV_IAR or GICV_AIAR returns a spurious interrupt ID.

This field resets to 0.

**Accessing the ICH_HCR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
ICH_HCR, Interrupt Controller Hyp Control Register

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_HCR;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_HCR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  else
    ICH_HCR = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICH_HCR = R[t];
ICH_LRC<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LRC<n> characteristics are:

**Purpose**

Provides interrupt context information for the virtual CPU interface.

**Configuration**

AArch32 System register ICH_LRC<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_LR<n>_EL2[63:32].

If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_LRC<n> is a 32-bit register.

**Field descriptions**

The ICH_LRC<n> bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>State</th>
<th>HW Group</th>
<th>RES0</th>
<th>Priority</th>
<th>RES0</th>
<th>pINTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State, bits [31:30]**

The state of the interrupt:

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Invalid (Inactive).</td>
</tr>
<tr>
<td>0b01</td>
<td>Pending.</td>
</tr>
<tr>
<td>0b10</td>
<td>Active.</td>
</tr>
<tr>
<td>0b11</td>
<td>Pending and active.</td>
</tr>
</tbody>
</table>

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

This field resets to 0.

**HW, bit [29]**

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the INTID that the pINTID field indicates.
<table>
<thead>
<tr>
<th>HW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated.</td>
</tr>
<tr>
<td>0b1</td>
<td>The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical INTID. If ICH_VMCR VEOIM is 0, this request corresponds to a write to ICC_EOIR0 or ICC_EOIR1. Otherwise, it corresponds to a write to ICC_DIR.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Group, bit [28]**

Indicates the group for this virtual interrupt.

<table>
<thead>
<tr>
<th>Group</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This is a Group 0 virtual interrupt. ICH_VMCR VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR VENG0 enables signaling of this interrupt to the virtual machine.</td>
</tr>
<tr>
<td>0b1</td>
<td>This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR VENG1 enables the signaling of this interrupt to the virtual machine. If ICH_VMCR VCBPR is 0, then ICC_BPR1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR&lt;n&gt; determines preemption.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Bits [27:24]**

Reserved, RES0.

**Priority, bits [23:16]**

The priority of this interrupt.

It is implementation defined how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[16] up to bit[18]. The number of implemented bits can be discovered from ICH_VTR.PRIbits.

This field resets to 0.

**Bits [15:13]**

Reserved, RES0.

**pINTID, bits [12:0]**

Physical INTID, for hardware interrupts.

When ICH_LRC<n>.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[12:10] : RES0.
- Bit[9] : EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits[8:0] : Reserved, RES0.

When ICH_LRC<n>.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
• When ICC_CTLR_EL1.ExtRange is 0, then bits[44:42] of this field are RES0.
• If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by ICC_CTLR.IDbits.

This field resets to 0.

**Accessing the ICH_LRC<n>**

ICH_LRC<n> and ICH_LRC<n> can be updated independently.

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b111:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elseif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elseself EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elseself PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
elseself PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
elseself
    return ICH_LRC[UInt(CRm<0>:opc2<2:0>)];
elseself PSTATE.EL == EL4 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
elseself
    return ICH_LRC[UInt(CRm<0>:opc2<2:0>)];

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b111:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end if;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC[UInt(CRm<0>:opc2<2:0>)] = R[t];
    end if;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC[UInt(CRm<0>:opc2<2:0>)] = R[t];
    end if;
ICH_LR<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n> characteristics are:

**Purpose**

Provides interrupt context information for the virtual CPU interface.

**Configuration**

AArch32 System register ICH_LR<n> bits [31:0] are architecturally mapped to AArch64 System register ICH_LR<n>_EL2[31:0].  
If EL2 is not implemented, this register is RES0 from EL3.  
If list register n is not implemented, then accesses to this register are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_LR<n> is a 32-bit register.

**Field descriptions**

The ICH_LR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    | vINTID |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**vINTID, bits [31:0]**

Virtual INTID of the interrupt.  
If the value of vINTID is 1020-1023 and ICH_LRC<n>.State!=0b00 (Inactive), behavior is UNPREDICTABLE.  
Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LRC<n>.State == 01.  
- ICH_LRC<n>.State == 10.  
- ICH_LRC<n>.State == 11.  

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from ICH_VTR.IDbits.

**Note**

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

This field resets to 0.

**Accessing the ICH_LR<n>**

ICH_LR<n> and ICH_LRC<n> can be updated independently.
Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1100:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LR[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LR[UInt(CRm<0>:opc2<2:0>)];

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1100:n[3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[UInt(CRm<0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[UInt(CRm<0>:opc2<2:0>)] = R[t];
ICH_MISR, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR characteristics are:

**Purpose**

Indicates which maintenance interrupts are asserted.

**Configuration**

AArch32 System register ICH_MISR bits [31:0] are architecturally mapped to AArch64 System register ICH_MISR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_MISR is a 32-bit register.

**Field descriptions**

The ICH_MISR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Number</th>
<th>Bit Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>VGrp1D</td>
<td>vPE Group 1 Disabled. Maintenance Interrupt asserted when ICH_HCR.VGrp1DIE==1 and ICH_VMCR.VMGrp1En==0. This field resets to 0.</td>
</tr>
<tr>
<td>6</td>
<td>VGrp1E</td>
<td>vPE Group 1 Enabled. Maintenance Interrupt asserted when ICH_HCR.VGrp1EIE==1 and ICH_VMCR.VMGrp1En==1. This field resets to 0.</td>
</tr>
</tbody>
</table>
VGrp0D, bit [5]

vPE Group 0 Disabled.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 0 Disabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 0 Disabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when ICH_HCR.VGrp0DIE==1 and ICH_VMCR.VMGrp0En==0. This field resets to 0.

VGrp0E, bit [4]

vPE Group 0 Enabled.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 0 Enabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 0 Enabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when ICH_HCR.VGrp0EIE==1 and ICH_VMCR.VMGrp0En==1. This field resets to 0.

NP, bit [3]

No Pending.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Pending maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>No Pending maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when ICH_HCR.NPIE==1 and no List register is in pending state. This field resets to 0.

LRENP, bit [2]

List Register Entry Not Present.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>List Register Entry Not Present maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>List Register Entry Not Present maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when ICH_HCR.LRENPIE==1 and ICH_HCR.EOICount is non-zero. This field resets to 0.

U, bit [1]

Underflow.

<table>
<thead>
<tr>
<th></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Underflow maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Underflow maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when ICH_HCR.UIE==1 and zero or one of the List register entries are marked as a valid interrupt, that is, if the corresponding ICH_LRC<n>.State bits do not equal 0x0. This field resets to 0.
**EOI, bit [0]**

End Of Interrupt.

<table>
<thead>
<tr>
<th>EOI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>End Of Interrupt maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>End Of Interrupt maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when at least one bit in `ICH_EISR` is 1.

This field resets to 0.

**Accessing the ICH_MISR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```java
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
else
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_MISR;
else
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_MISR;
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ICH_VMCR, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR characteristics are:

**Purpose**

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

**Configuration**

AArch32 System register ICH_VMCR bits [31:0] are architecturally mapped to AArch64 System register ICH_VMCR_EL2[31:0].

If EL2 is not implemented, this register is RES0 from EL3.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICH_VMCR is a 32-bit register.

**Field descriptions**

The ICH_VMCR bit assignments are:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 | VPMR VBPR0 VBPR1 RES0 VEOIM RES0 VCBPR VFIQEn VAckCtl VENG1 VENG0 |
|----------------------------------------------------------------------------------------------------------------------------------|

**VPMR, bits [31:24]**

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of ICV_PMR.Priority.

This field resets to an architecturally UNKNOWN value.

**VBPR0, bits [23:21]**

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR.VCBPR == 1.

This field is an alias of ICV_BPR0.BinaryPoint.

This field resets to an architecturally UNKNOWN value.

**VBPR1, bits [20:18]**

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if ICH_VMCR.VCBPR == 0.

This field is an alias of ICV_BPR1.BinaryPoint.

This field resets to an architecturally UNKNOWN value.
Bits [17:10]

Reserved, RES0.

**VEOIM, bit [9]**

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

<table>
<thead>
<tr>
<th>VEOIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This bit is an alias of ICV_CTLR.EOImode.

This field resets to an architecturally UNKNOWN value.

**Bits [8:5]**

Reserved, RES0.

**VCBPR, bit [4]**

Virtual Common Binary Point Register. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VCBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only. ICV_BPR1 determines the preemption group for virtual Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>ICV_BPR0 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts. Reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 0b111. Writes to ICV_BPR1 are ignored.</td>
</tr>
</tbody>
</table>

This field is an alias of ICV_CTLR.CBPR.

This field resets to an architecturally UNKNOWN value.

**VFIQEn, bit [3]**

Virtual FIQ enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VFIQEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 virtual interrupts are presented as virtual IRQs.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 virtual interrupts are presented as virtual FIQs.</td>
</tr>
</tbody>
</table>

This bit is an alias of GICV_CTLR.FIQEn.

In implementations where the Non-secure copy of ICC_SRE.SRE is always 1, this bit is RES1.

This field resets to an architecturally UNKNOWN value.

**VAckCtl, bit [2]**

Virtual AckCtl. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VAckCtl</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt.</td>
</tr>
</tbody>
</table>
This bit is an alias of GICV_CTLR.AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of ICC_SRE.SRE is always 1, this bit is \texttt{RES0}.

This field resets to an architecturally \texttt{UNKNOWN} value.

**VENG1, bit [1]**

Virtual Group 1 interrupt enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VENG1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This bit is an alias of ICV_IGRPEN1.Enable.

This field resets to an architecturally \texttt{UNKNOWN} value.

**VENG0, bit [0]**

Virtual Group 0 interrupt enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VENG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This bit is an alias of ICV_IGRPEN0.Enable.

This field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the \texttt{ICH_VMCR}**

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>\{,#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{,#\}<\text{opc2}> \\
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{coproc} & \text{opc1} & \text{CRn} & \text{CRm} & \text{opc2} \\
\hline
0b1111 & 0b100 & 0b1100 & 0b1011 & 0b111 \\
\hline
\end{array}
\]

if \text{PSTATE.EL} == \text{EL0} then \textbf{UNDEFINED};

elsif \text{PSTATE.EL} == \text{EL1} then
  if \text{EL2Enabled()} && \text{ELUsingAArch32(EL2)} && \text{HSTR.EL2.T12} == '1' then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \text{EL2Enabled()} && \text{ELUsingAArch32(EL2)} && \text{HSTR.T12} == '1' then
    \text{AArch32.TakeHypTrapException(0x03)};
  else
    \textbf{UNDEFINED};
  endif
else
  \textbf{UNDEFINED};
endif

elsif \text{PSTATE.EL} == \text{EL2} then
  if \text{ICC_HSRE.SRE} == '0' then
    \textbf{UNDEFINED};
  else
    return \text{ICH_VMCR};
  endif
else
  \textbf{UNDEFINED};
endif

elsif \text{PSTATE.EL} == \text{EL3} then
  if \text{ICC_MSRE.SRE} == '0' then
    \textbf{UNDEFINED};
  else
    return \text{ICH_VMCR};
  endif
else
  \textbf{UNDEFINED};
endif
ICH_VMCR, Interrupt Controller Virtual Machine Control Register

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_VMCR = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_VMCR = R[t];
    endif
endif
ICH_VTR, Interrupt Controller VGIC Type Register

The ICH_VTR characteristics are:

**Purpose**

Reports supported GIC virtualisation features.

**Configuration**

AArch32 System register ICH_VTR bits [31:0] are architecturally mapped to AArch64 System register ICH_VTR_EL2[31:0].

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

**Attributes**

ICH_VTR is a 32-bit register.

**Field descriptions**

The ICH_VTR bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIbits</td>
<td>Priority bits. The number of virtual priority bits implemented, minus one. An implementation must implement at least 32 levels of virtual priority (5 priority bits). This field is an alias of ICV_CTLR.PRIbits.</td>
</tr>
<tr>
<td>PREbits</td>
<td>The number of virtual preemption bits implemented, minus one. An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits). The value of this field must be less than or equal to the value of ICH_VTR.PRIbits.</td>
</tr>
<tr>
<td>IDbits</td>
<td>The number of virtual interrupt identifier bits supported:</td>
</tr>
<tr>
<td>SEIS</td>
<td>SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:</td>
</tr>
</tbody>
</table>

PRIbits, bits [31:29]

Preemption bits. The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

This field is an alias of ICV_CTLR.PRIbits.

PREbits, bits [28:26]

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR.PRIbits.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is an alias of ICV_CTLR.IDbits.

SEIS, bit [22]

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:
<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic does not support generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports generation of SEIs.</td>
</tr>
</tbody>
</table>

This bit is an alias of `ICV_CTLR.SEIS`.

### A3V, bit [21]

**Affinity 3 Valid. Possible values are:**

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
</tbody>
</table>

This bit is an alias of `ICV_CTLR.A3V`.

### nV4, bit [20]

**Direct injection of virtual interrupts not supported. Possible values are:**

<table>
<thead>
<tr>
<th>nV4</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The CPU interface logic supports direct injection of virtual interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>The CPU interface logic does not support direct injection of virtual interrupts.</td>
</tr>
</tbody>
</table>

In GICv3 this bit is `RES1`.

### TDS, bit [19]

**Separate trapping of Non-secure EL1 writes to `ICV_DIR` supported.**

<table>
<thead>
<tr>
<th>TDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Implementation does not support <code>ICH_HCR.TDIR</code>.</td>
</tr>
<tr>
<td>0b1</td>
<td>Implementation supports <code>ICH_HCR.TDIR</code>.</td>
</tr>
</tbody>
</table>

### Bits [18:5]

Reserved, `RES0`.

### ListRegs, bits [4:0]

The number of implemented List registers, minus one. For example, a value of 0b01111 indicates that the maximum of 16 List registers are implemented.

### Accessing the ICH_VTR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_VTR;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICH_VTR;

ICIALLU, Instruction Cache Invalidate All to PoU

The ICIALLU characteristics are:

**Purpose**

Invalidate all instruction caches to PoU. If branch predictors are architecturally visible, also flush branch predictors.

**Configuration**

AArch32 System instruction ICIALLU performs the same function as AArch64 System instruction IC IALLU.

**Attributes**

ICIALLU is a 32-bit System instruction.

**Field descriptions**

ICIALLU ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the ICIALLU instruction**

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

When HCR.FB is 1, at Non-secure EL1 this instruction executes as a ICIALLUIS.

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TPU == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
    ICIALLUIS();
else
  ICIALLU();
elsif PSTATE.EL == EL2 then
  ICIALLU();
elsif PSTATE.EL == EL3 then
  ICIALLU();
ICIALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable

The ICIALLUIS characteristics are:

**Purpose**

Invalidate all instruction caches Inner Shareable to PoU. If branch predictors are architecturally visible, also flush branch predictors.

**Configuration**

AArch32 System instruction ICIALLUIS performs the same function as AArch64 System instruction IC IALLUIS.

**Attributes**

ICIALLUIS is a 32-bit System instruction.

**Field descriptions**

ICIALLUIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the ICIALLUIS instruction**

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TICAB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TICAB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ICIALLU();
    end
elsif PSTATE.EL == EL2 then
    ICIALLU();
elsif PSTATE.EL == EL3 then
    ICIALLU();
ICIMVAU, Instruction Cache line Invalidate by VA to PoU

The ICIMVAU characteristics are:

**Purpose**
Invalidate instruction cache line by virtual address to PoU.

**Configuration**
AArch32 System instruction ICIMVAU performs the same function as AArch64 System instruction IC IVAU.

**Attributes**
ICIMVAU is a 32-bit System instruction.

**Field descriptions**
The ICIMVAU input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Virtual address to use**

**Bits [31:0]**
Virtual address to use. No alignment restrictions apply to this VA.

**Executing the ICIMVAU instruction**
Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see ‘AArch32 instruction cache maintenance instruction (IC*)’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this instruction use the following encodings:

\[
\text{MCR} \{<c>\}{<q>} \text{ <coproc>}, \{#<opc1>\}, \text{ <Rt>}, \text{ <CRn>}, \text{<CRm>}, \{#<opc2>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
    AArch64.AArch32SystemAccessTrap(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
    AArch32.TakeHypTrapException(0x03);
else
  ICIMVAU(R[t]);
elsif PSTATE.EL == EL2 then
  ICIMVAU(R[t]);
eelsif PSTATE.EL == EL3 then
  ICIMVAU(R[t]);
The ICV_AP0R\(n\) characteristics are:

### Purpose

Provides information about virtual Group 0 active priorities.

### Configuration

AArch32 System register ICV_AP0R\(n\) bits [31:0] are architecturally mapped to AArch64 System register ICV_AP0R\(n\)_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally \texttt{UNKNOWN} values.

### Attributes

ICV_AP0R\(n\) is a 32-bit register.

### Field descriptions

The ICV_AP0R\(n\) bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\texttt{IMPLEMENTATION DEFINED}

**IMPLEMENTATION DEFINED, bits [31:0]**

\texttt{IMPLEMENTATION DEFINED}.

This field resets to 0.

The contents of these registers are \texttt{IMPLEMENTATION DEFINED} with the one architectural requirement that the value \texttt{0x00000000} is consistent with no interrupts being active.

### Accessing the ICV_AP0R\(n\)

Writing to these registers with any value other than the last read value of the register (or \texttt{0x00000000} when there are no Group 0 active priorities) might result in \texttt{UNPREDICTABLE} behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP0R1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP0R2 and ICV_AP0R3 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are \texttt{UNDEFINED}.

Writing to the active priority registers in any order other than the following order might result in \texttt{UNPREDICTABLE} behavior of the interrupt prioritization system:

- ICV_AP0R\(n\).
- ICV_AP1R\(n\).

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif ICC_HSRE.SRE == '0' then
      UNDEFINED;
elif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
      AArch32.TakeHypTrapException(0x03);
elif ICC_MSRE.SRE == '0' then
      UNDEFINED;
elif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elif ICC_MSRE.SRE == '0' then
      UNDEFINED;
elif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
      return ICV_AP0R[UInt(opc2<1:0>)];
elif ICC_MSRE.SRE == '0' then
      UNDEFINED;
elif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR_EL2.FMO == '1' then
      return ICV_AP0R[UInt(opc2<1:0>)];
elif ICC_MSRE.SRE == '0' then
      UNDEFINED;
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    endif
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    endif

The ICV_AP1R<n> characteristics are:

**Purpose**

Provides information about virtual Group 1 active priorities.

**Configuration**

AArch32 System register ICV_AP1R<n> bits [31:0] are architecturally mapped to AArch64 System register ICV_AP1R<n>_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICV_AP1R<n> is a 32-bit register.

**Field descriptions**

The ICV_AP1R<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| **IMPLEMENTATION DEFINED** |

**IMPLEMENTATION DEFINED, bits [31:0]**

**IMPLEMENTATION DEFINED.**

This field resets to 0.

The contents of these registers are **IMPLEMENTATION DEFINED** with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

**Accessing the ICV_AP1R<n>**

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in **UNPREDICTABLE** behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP1R1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP1R2 and ICV_AP1R3 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are **UNDEFINED**.

Writing to the active priority registers in any order other than the following order might result in **UNPREDICTABLE** behavior of the interrupt prioritization system:

- ICV_AP0R<n>.
- ICV_AP1R<n>.

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_AP1R[UInt(opc2<1:0>)];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_AP1R[UInt(opc2<1:0>)];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        if SCR.NS == '0' then
            return ICC_AP1R_S[UInt(opc2<1:0>)];
        else
            return ICC_AP1R_NS[UInt(opc2<1:0>)];
        end
    else
        return ICC_AP1R[UInt(opc2<1:0>)];
    end
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_AP1R_NS[UInt(opc2<1:0>)];
    else
        return ICC_AP1R[UInt(opc2<1:0>)];
    end
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_AP1R_S[UInt(opc2<1:0>)];
        else
            return ICC_AP1R_NS[UInt(opc2<1:0>)];
        end
    end
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHyp TrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICC_AP1R[UInt(opc2<1:0>)] = R[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
elsif HaveEL(EL3) then
  if SCR.NS == '0' then
    ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
  else
    ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
else
  ICC_AP1R[UInt(opc2<1:0>)] = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
elsif HaveEL(EL3) then
  ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
else
  ICC_AP1R[UInt(opc2<1:0>)] = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    if SCR.NS == '0' then
      ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
    else
      ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
The ICV_BPR0 characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

**Configuration**

AArch32 System register ICV_BPR0 bits [31:0] are architecturally mapped to AArch64 System register ICV_BPR0_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_BPR0 is a 32-bit register.

**Field descriptions**

The ICV_BPR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BinaryPoint |

**Bits [31:3]**

Reserved, RES0.

**BinaryPoint, bits [2:0]**

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

<table>
<thead>
<tr>
<th>Binary point value</th>
<th>Group priority field</th>
<th>Subpriority field</th>
<th>Field with binary point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[7:1]</td>
<td>[0]</td>
<td>ggggggg.s</td>
</tr>
<tr>
<td>1</td>
<td>[7:2]</td>
<td>[1:0]</td>
<td>gggggg.ss</td>
</tr>
<tr>
<td>2</td>
<td>[7:3]</td>
<td>[2:0]</td>
<td>gggggg.sss</td>
</tr>
<tr>
<td>3</td>
<td>[7:4]</td>
<td>[3:0]</td>
<td>ggggg.sssss</td>
</tr>
<tr>
<td>4</td>
<td>[7:5]</td>
<td>[4:0]</td>
<td>ggg.sssssss</td>
</tr>
<tr>
<td>5</td>
<td>[7:6]</td>
<td>[5:0]</td>
<td>gg.sssssss</td>
</tr>
<tr>
<td>6</td>
<td>[7]</td>
<td>[6:0]</td>
<td>g.sssssssss</td>
</tr>
<tr>
<td>7</td>
<td>No preemption</td>
<td>[7:0]</td>
<td>.ssssssssss</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the ICV_BPR0**

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by ICV_CTLR PRibits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is set to the minimum supported value.
Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{<\text{opc2}>\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if \(\text{PSTATE.EL} == \text{EL0}\) then
  \text{UNDEFINED};
elsif \(\text{PSTATE.EL} == \text{EL1}\) then
  if \(\text{EL2Enabled()} \&\& \text{!ELUsingAArch32(EL2)} \&\& \text{HSTR\_EL2.T12} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HSTR\_T12} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \(\text{ICC\_SRE.SRE} == '0'\) then
    \text{UNDEFINED};
  elsif \(\text{EL2Enabled()} \&\& \text{!ELUsingAArch32(EL2)} \&\& \text{ICH\_HCR\_EL2.TALL0} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{ICH\_HCR.TALL0} == '1'\) then
    \text{AArch32.TakeHypTrapException(0x03)};
  elsif \(\text{EL2Enabled()} \&\& \text{!ELUsingAArch32(EL2)} \&\& \text{HCR\_EL2.FMO} == '1'\) then
    \text{return ICV\_BPR0};
  elsif \(\text{EL2Enabled()} \&\& \text{ELUsingAArch32(EL2)} \&\& \text{HCR.FMO} == '1'\) then
    \text{return ICV\_BPR0};
  elsif \(\text{HaveEL(EL3)} \&\& \text{!ELUsingAArch32(EL3)} \&\& \text{SCR\_EL3.FIQ} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  elsif \(\text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{ SCR\_FIQ} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  else
    \text{return ICC\_BPR0};
  endforeach
elsif \(\text{PSTATE.EL} == \text{EL2}\) then
  if \(\text{ICC\_HSRE.SRE} == '0'\) then
    \text{UNDEFINED};
  elsif \(\text{HaveEL(EL3)} \&\& \text{!ELUsingAArch32(EL3)} \&\& \text{SCR\_EL3.FIQ} == '1'\) then
    \text{AArch64.AArch32SystemAccessTrap(EL3, 0x03)};
  elsif \(\text{HaveEL(EL3)} \&\& \text{ELUsingAArch32(EL3)} \&\& \text{SCR\_FIQ} == '1'\) then
    \text{AArch32.TakeMonitorTrapException()};
  else
    \text{return ICC\_BPR0};
  endforeach
elsif \(\text{PSTATE.EL} == \text{EL3}\) then
  if \(\text{ICC\_MSRE.SRE} == '0'\) then
    \text{UNDEFINED};
  else
    \text{return ICC\_BPR0};
  endforeach
endmodule
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH.HCR.TALL0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICV_BPR0 = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    ICV_BPR0 = R[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  end;
elsif ICC_BPR0 = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    ICC_BPR0 = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICC_BPR0 = R[t];
The ICV_BPR1 characteristics are:

**Purpose**

 Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

**Configuration**

AArch32 System register ICV_BPR1 bits [31:0] are architecturally mapped to AArch64 System register ICV_BPR1_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally **Unknown** values.

**Attributes**

ICV_BPR1 is a 32-bit register.

**Field descriptions**

The ICV_BPR1 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RES0</td>
<td>BinaryPoint</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**BinaryPoint, bits [2:0]**

If the GIC is configured to use separate binary point fields for virtual Group 0 and virtual Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

<table>
<thead>
<tr>
<th>Binary point value</th>
<th>Group priority field</th>
<th>Subpriority field</th>
<th>Field with binary point</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1</td>
<td>[7:1]</td>
<td>[0]</td>
<td>ggggggg.g</td>
</tr>
<tr>
<td>2</td>
<td>[7:2]</td>
<td>[1:0]</td>
<td>gggggg.ss</td>
</tr>
<tr>
<td>3</td>
<td>[7:3]</td>
<td>[2:0]</td>
<td>gggggg.sss</td>
</tr>
<tr>
<td>4</td>
<td>[7:4]</td>
<td>[3:0]</td>
<td>ggggsss</td>
</tr>
<tr>
<td>5</td>
<td>[7:5]</td>
<td>[4:0]</td>
<td>ggg.sssss</td>
</tr>
<tr>
<td>6</td>
<td>[7:6]</td>
<td>[5:0]</td>
<td>gg.ssssss</td>
</tr>
<tr>
<td>7</td>
<td>[7]</td>
<td>[6:0]</td>
<td>g.sssssss</td>
</tr>
</tbody>
</table>

Writing 0 to this field will set this field to its reset value.

If ICV_CTLR CBPR is set to 1, Non-secure EL1 reads return ICV_BPR0 + 1 saturated to 0b111. Non-secure EL1 writes are ignored.

This field resets to an IMPLEMENTATION DEFINED non-zero value.
Accessing the ICV_BPR1

The reset value is `IMPLEMENTATION DEFINED`, but is equal to the minimum value of `ICV_BPR0` plus one.

An attempt to program the binary point field to a value less than the reset value sets the field to the reset value.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
  return ICV_BPR1;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
  return ICV_BPR1;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE_M != M32_Monitor && SCR.IRQ == '1' then
  AArch32.TakeMonitorTrapException();
elsif HaveEL(EL3) then
  if SCR.NS == '0' then
    return ICC_BPR1_S;
  else
    return ICC_BPR1_NS;
else
  return ICC_BPR1;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
  AArch32.TakeMonitorTrapException();
elsif HaveEL(EL3) then
  return ICC_BPR1_NS;
else
  return ICC_BPR1;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  if SCR.NS == '0' then
    return ICC_BPR1_S;
  else
    return ICC_BPR1_NS;

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif ICV_BPR1.IMO == '1' then
    ICV_BPR1 = R[t];
  elsif ICC_HCR_EL2.IMO == '1' then
    ICC_BPR1_NS = R[t];
else
  ICC_BPR1 = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif ICC_HCR_EL2.IMO == '1' then
    ICV_BPR1 = R[t];
else
  ICC_BPR1_NS = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  ICC_BPR1 = R[t];
elsif PSTATE.EL == EL4 then
  if SCR.NS == '0' then
    ICC_BPR1_S = R[t];
else
  ICC_BPR1_NS = R[t];
ICV_CTLR, Interrupt Controller Virtual Control Register

The ICV_CTLR characteristics are:

**Purpose**

Controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

**Configuration**

AArch32 System register ICV_CTLR bits [31:0] are architecturally mapped to AArch64 System register ICV_CTLR_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_CTLR is a 32-bit register.

**Field descriptions**

The ICV_CTLR bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | ExtRange | RSS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | EOImode | CBPR |

**Bits [31:20]**

Reserved, RES0.

**ExtRange, bit [19]**

Extended INTID range (read-only).

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>CPU interface does not support INTIDs in the range 1024..8191. Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.</td>
</tr>
</tbody>
</table>

**Note**

Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.

<table>
<thead>
<tr>
<th>ExtRange</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation.</td>
</tr>
</tbody>
</table>

ICV_CTLR.ExtRange is an alias of ICC_CTLR.ExtRange.

**RSS, bit [18]**

Range Selector Support. Possible values are:
**RSS**

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 15 are supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>Targeted SGIs with affinity level 0 values of 0 - 255 are supported.</td>
</tr>
</tbody>
</table>

This bit is read-only.

**Bits [17:16]**

Reserved, RES0.

**A3V, bit [15]**

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers.</td>
</tr>
</tbody>
</table>

**SEIS, bit [14]**

SEI Support. Read-only and writes are ignored. Indicates whether the virtual CPU interface supports local generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic does not support local generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports local generation of SEIs.</td>
</tr>
</tbody>
</table>

**IDbits, bits [13:11]**

Identifier bits. Read-only and writes are ignored. The number of virtual interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>0b001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**PRIbits, bits [10:8]**

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation must implement at least 32 levels of physical priority (5 priority bits).

**Note**

This field always returns the number of priority bits implemented.

The division between group priority and subpriority is defined in the binary point registers `ICV_BPR0` and `ICV_BPR1`.

**Bits [7:2]**

Reserved, RES0.

**EOImode, bit [1]**

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:
**EOImode**

<table>
<thead>
<tr>
<th>EOImode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>ICV_EOIR0</strong> and <strong>ICV_EOIR1</strong> provide both priority drop and interrupt deactivation functionality. Accesses to <strong>ICV_DIR</strong> are <strong>UNPREDICTABLE</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>ICV_EOIR0</strong> and <strong>ICV_EOIR1</strong> provide priority drop functionality only. <strong>ICV_DIR</strong> provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**CBPR, bit [0]**

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts:

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>ICV_BPR0</strong> determines the preemption group for virtual Group 0 interrupts only. <strong>ICV_BPR1</strong> determines the preemption group for virtual Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>ICV_BPR0</strong> determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts. Reads of <strong>ICV_BPR1</strong> return <strong>ICV_BPR0</strong> plus one, saturated to 0b111. Writes to <strong>ICV_BPR1</strong> are ignored.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Accessing the ICV_CTLR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c}\{<q}\} <\text{coproc}, \{#<\text{opc1}\}, <\text{Rt}> , <\text{CRn}>, <\text{CRm}>\{, \{#<\text{opc2}\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch32.AArch64SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    AArch32.AArch64SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_CTLR;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    return ICV_CTLR;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    return ICV_CTLR;
  elsif EL2Enabled() && ELUsingAArch32(EL2) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    if SCR.NS == '0' then
      return ICC_CTLR_S;
    else
      return ICC_CTLR_NS;
    end if
  else
    return ICC_CTLR;
  end if
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    return ICC_CTLR_NS;
  else
    return ICC_CTLR;
  end if
elsif PSTATE.EL == EL3 then
  if ICC_MSRRE.SRE == '0' then
    UNDEFINED;
  else
    if SCR.NS == '0' then
      return ICC_CTLR_S;
    else
      return ICC_CTLR_NS;
    end if
  end if
end if

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    if SCR.NS == '0' then
      ICC_CTLR_S = R[t];
    else
      ICC_CTLR_NS = R[t];
    end if;
  else
    ICC_CTLR = R[t];
  end if;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif HaveEL(EL3) then
    ICC_CTLR_NS = R[t];
  else
    ICC_CTLR = R[t];
  end if;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    if SCR.NS == '0' then
      ICC_CTLR_S = R[t];
    else
      ICC_CTLR_NS = R[t];
  end if;
else
  ICC_CTLR = R[t];
end if;
ICV_DIR, Interrupt Controller Deactivate Virtual Interrupt Register

The ICV_DIR characteristics are:

**Purpose**

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified virtual interrupt.

**Configuration**

AArch32 System register ICV_DIR bits [31:0] performs the same function as AArch64 System register ICV_DIR_EL1[31:0].

**Attributes**

ICV_DIR is a 32-bit register.

**Field descriptions**

The ICV_DIR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>RES0</td>
</tr>
<tr>
<td>23-0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the virtual interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_DIR**

When EOImode == 0, writes are ignored. In systems supporting system error generation, an implementation might generate an SEI.

Accesses to this register use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TDIR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TDIR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICV_DIR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICV_DIR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    ICV_DIR = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    ICV_DIR = R[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  else
    ICC_DIR = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  else
    ICC_DIR = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICC_DIR = R[t];
ICV_EOIR0, Interrupt Controller Virtual End Of Interrupt Register 0

The ICV_EOIR0 characteristics are:

**Purpose**

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 0 interrupt.

**Configuration**

AArch32 System register ICV_EOIR0 performs the same function as AArch64 System register ICV_EOIR0_EL1.

**Attributes**

ICV_EOIR0 is a 32-bit register.

**Field descriptions**

The ICV_EOIR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23-0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID from the corresponding ICV_IAR0 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR to deactivate the virtual interrupt.

**Accessing the ICV_EOIR0**

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR0, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH.HCR.TALL0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    ICV_EOIR0 = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    ICV_EOIR0 = R[t];
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
else
  ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
else
  ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  ICC_EOIR0 = R[t];
ICV_EOIR1, Interrupt Controller Virtual End Of Interrupt Register 1

The ICV_EOIR1 characteristics are:

**Purpose**

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 1 interrupt.

**Configuration**

AArch32 System register ICV_EOIR1 performs the same function as AArch64 System register ICV_EOIR1_EL1.

**Attributes**

ICV_EOIR1 is a 32-bit register.

**Field descriptions**

The ICV_EOIR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit pos</th>
<th>Bit assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>INTID</td>
<td>The INTID from the corresponding ICV_IAR1 access.</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID from the corresponding ICV_IAR1 access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the ICV_CTLR.EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the ICV_CTLR.EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to ICV_DIR to deactivate the virtual interrupt.

**Accessing the ICV_EOIR1**

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from ICV_IAR1, otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_TALL1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    ICV_EOIR1 = R[t];
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
else
  ICC_EOIR1 = R[t];
endif
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
else
  ICC_EOIR1 = R[t];
endif
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    ICC_EOIR1 = R[t];
  endif
ICV_HPPIR0, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

The ICV_HPPIR0 characteristics are:

**Purpose**

Indicates the highest priority pending virtual Group 0 interrupt on the virtual CPU interface.

**Configuration**

AArch32 System register ICV_HPPIR0 performs the same function as AArch64 System register [ICV_HPPIR0_EL1](ICV_HPPIR0_EL1).

**Attributes**

ICV_HPPIR0 is a 32-bit register.

**Field descriptions**

The ICV_HPPIR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](ICV_CTLR.IDbits). If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_HPPIR0**

Accesses to this register use the following encodings:

\[ \text{MRC} \{<c>\}{<q>} \text{<coproc>, \{<opc1>, <CRn>, <CRm>{, \{<opc2>}} \}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_HPPIR0;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    return ICV_HPPIR0;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_HPPIR0;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_HPPIR0;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICC_HPPIR0;
ICV_HPPIR1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

The ICV_HPPIR1 characteristics are:

**Purpose**

Indicates the highest priority pending virtual Group 1 interrupt on the virtual CPU interface.

**Configuration**

AArch32 System register ICV_HPPIR1 performs the same function as AArch64 System register ICV_HPPIR1_EL1.

**Attributes**

ICV_HPPIR1 is a 32-bit register.

**Field descriptions**

The ICV_HPPIR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RESERVED</td>
</tr>
<tr>
<td>30</td>
<td>RESERVED</td>
</tr>
<tr>
<td>29</td>
<td>RESERVED</td>
</tr>
<tr>
<td>28</td>
<td>RESERVED</td>
</tr>
<tr>
<td>27</td>
<td>RESERVED</td>
</tr>
<tr>
<td>26</td>
<td>RESERVED</td>
</tr>
<tr>
<td>25</td>
<td>RESERVED</td>
</tr>
<tr>
<td>24</td>
<td>RESERVED</td>
</tr>
<tr>
<td>23</td>
<td>RESERVED</td>
</tr>
<tr>
<td>22</td>
<td>RESERVED</td>
</tr>
<tr>
<td>21</td>
<td>RESERVED</td>
</tr>
<tr>
<td>20</td>
<td>RESERVED</td>
</tr>
<tr>
<td>19</td>
<td>RESERVED</td>
</tr>
<tr>
<td>18</td>
<td>RESERVED</td>
</tr>
<tr>
<td>17</td>
<td>RESERVED</td>
</tr>
<tr>
<td>16</td>
<td>RESERVED</td>
</tr>
<tr>
<td>15</td>
<td>RESERVED</td>
</tr>
<tr>
<td>14</td>
<td>RESERVED</td>
</tr>
<tr>
<td>13</td>
<td>RESERVED</td>
</tr>
<tr>
<td>12</td>
<td>RESERVED</td>
</tr>
<tr>
<td>11</td>
<td>RESERVED</td>
</tr>
<tr>
<td>10</td>
<td>RESERVED</td>
</tr>
<tr>
<td>9</td>
<td>RESERVED</td>
</tr>
<tr>
<td>8</td>
<td>RESERVED</td>
</tr>
<tr>
<td>7</td>
<td>RESERVED</td>
</tr>
<tr>
<td>6</td>
<td>RESERVED</td>
</tr>
<tr>
<td>5</td>
<td>RESERVED</td>
</tr>
<tr>
<td>4</td>
<td>RESERVED</td>
</tr>
<tr>
<td>3</td>
<td>RESERVED</td>
</tr>
<tr>
<td>2</td>
<td>RESERVED</td>
</tr>
<tr>
<td>1</td>
<td>RESERVED</td>
</tr>
<tr>
<td>0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_HPPIR1**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>}} \text{ <coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>, \{#<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif ICC_SRE.SRE == '0' then
    UNDEFINED;
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_HPPIR1;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    return ICV_HPPIR1;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_HPPIR1;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
  else
    return ICC_HPPIR1;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    return ICC_HPPIR1;
ICV_IAR0, Interrupt Controller Virtual Interrupt Acknowledge Register 0

The ICV_IAR0 characteristics are:

**Purpose**

The PE reads this register to obtain the INTID of the signaled virtual Group 0 interrupt. This read acts as an acknowledge for the interrupt.

**Configuration**

AArch32 System register ICV_IAR0 performs the same function as AArch64 System register ICV_IAR0_EL1.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

**Attributes**

ICV_IAR0 is a 32-bit register.

**Field descriptions**

The ICV_IAR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

**Accessing the ICV_IAR0**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\{<q>\} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>\{, \{#\}<opc2>\}}
\]
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_IAR0;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_IAR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_IAR0;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return ICC_IAR0;
else
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_IAR0;
ICV_IAR1, Interrupt Controller Virtual Interrupt Acknowledge Register 1

The ICV_IAR1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICV_IAR1 performs the same function as AArch64 System register ICV_IAR1_EL1.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when PSTATE.{I,F} == {0,0}). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. See Observability of the effects of accesses to the GIC registers, for more information.

Attributes

ICV_IAR1 is a 32-bit register.

Field descriptions

The ICV_IAR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. See Special INTIDs, for more information.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in ICV_CTLR.IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing the ICV_IAR1

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

coproc | opc1 | CRn | CRm | opc2
|-------|------|-----|-----|------|

Page 2700
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
  return ICV_IAR1;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
  return ICV_IAR1;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
  AArch32.TakeMonitorTrapException();
else
  return ICC_IAR1;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
  AArch32.TakeMonitorTrapException();
else
  return ICC_IAR1;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  return ICC_IAR1;
ICV_IGRPEN0, Interrupt Controller Virtual Interrupt Group 0 Enable register

The ICV_IGRPEN0 characteristics are:

**Purpose**

Controls whether virtual Group 0 interrupts are enabled or not.

**Configuration**

AArch32 System register ICV_IGRPEN0 bits [31:0] are architecturally mapped to AArch64 System register `ICV_IGRPEN0_EL1[31:0]`.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_IGRPEN0 is a 32-bit register.

**Field descriptions**

The ICV_IGRPEN0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Enable</td>
<td>Enables virtual Group 0 interrupts.</td>
</tr>
</tbody>
</table>

**Bits [31:1]**

Reserved, RES0.

**Enable, bit [0]**

Enables virtual Group 0 interrupts.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Accessing the ICV_IGRPEN0**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then 
    UNDEFINED;
elsif PSTATE.EL == EL1 then 
    if EL2Enabled() && HSTR_EL2.T12 == '1' then 
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then 
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then 
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then 
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then 
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then 
        UNDEFINED;
else 
    return ICC_IGRPEN0;
elsif PSTATE.EL == EL2 then 
    if ICC_HSRE.SRE == '0' then 
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then 
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then 
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then 
        AArch32.TakeMonitorTrapException();
    else 
        return ICC_IGRPEN0;
elsif PSTATE.EL == EL3 then 
    if ICC_MSRE.SRE == '0' then 
        UNDEFINED;
    else 
        return ICC_IGRPEN0;
else 
    return ICC_IGRPEN0;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
elsif ICC_SRE.SRE == '0' then
    UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_IGRPEN0 = R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_IGRPEN0 = R[t];
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ICC_IGRPEN0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_IGRPEN0 = R[t];
ICV_IGRPEN1, Interrupt Controller Virtual Interrupt Group 1 Enable register

The ICV_IGRPEN1 characteristics are:

**Purpose**

Controls whether virtual Group 1 interrupts are enabled for the current Security state.

**Configuration**

AArch32 System register ICV_IGRPEN1 bits [31:0] are architecturally mapped to AArch64 System register ICV_IGRPEN1_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

ICV_IGRPEN1 is a 32-bit register.

**Field descriptions**

The ICV_IGRPEN1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>30</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>29</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>28</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>27</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>26</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>25</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>24</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>23</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>22</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>21</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>20</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>19</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>18</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>17</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>16</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>15</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>14</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>13</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>12</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>11</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>10</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>9</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>8</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>7</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>6</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>5</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>4</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>3</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>2</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>1</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>0</td>
<td>Enable</td>
</tr>
</tbody>
</table>

**Bits [31:1]**

Reserved, **RES0**.

**Enable, bit [0]**

Enables virtual Group 1 interrupts.

<table>
<thead>
<tr>
<th>Enable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Accessing the ICV_IGRPEN1**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif ICC SRE.SRE == '0' then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_IGRPEN1;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    return ICV_IGRPEN1;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
  AArch32.TakeMonitorTrapException();
elsif HaveEL(EL3) then
  if SCR.NS == '0' then
    return ICC_IGRPEN1_S;
  else
    return ICC_IGRPEN1_NS;
else
  return ICC_IGRPEN1;
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
    AArch32.TakeMonitorTrapException();
elsif HaveEL(EL3) then
  return ICC_IGRPEN1_NS;
else
  return ICC_IGRPEN1;
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
  else
    if SCR.NS == '0' then
      return ICC_IGRPEN1_S;
    else
      return ICC_IGRPEN1_NS;
else
  return ICC_IGRPEN1;

copcroc{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif ICC_SRE.SRE == '0' then
  UNDEFINED;
elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
  ICC_IGRPEN1 = R[t];
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
  ICC_IGRPEN1 = R[t];
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
  AArch32.TakeMonitor TrapException();
elsif HaveEL(EL3) then
  if SCR.NS == '0' then
    ICC_IGRPEN1_S = R[t];
  else
    ICC_IGRPEN1_NS = R[t];
else
  ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
  AArch32.TakeMonitor TrapException();
elsif HaveEL(EL3) then
  ICC_IGRPEN1_NS = R[t];
else
  ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  if SCR.NS == '0' then
    ICC_IGRPEN1_S = R[t];
  else
    ICC_IGRPEN1_NS = R[t];

The ICV_PMR characteristics are:

**Purpose**

Provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

**Configuration**

AArch32 System register ICV_PMR bits [31:0] are architecturally mapped to AArch64 System register ICV_PMR_EL1[31:0].

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. See Observability of the effects of accesses to the GIC registers, for more information.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

ICV_PMR is a 32-bit register.

**Field descriptions**

The ICV_PMR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>Priority bits [31:8] Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The priority mask level for the virtual CPU interface. If the priority of a virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

The possible priority field values are as follows:

<table>
<thead>
<tr>
<th>Implemented priority bits</th>
<th>Possible priority field values</th>
<th>Number of priority levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>[7:0]</td>
<td>0x00-0xFF (0-255), all values</td>
<td>256</td>
</tr>
<tr>
<td>[7:1]</td>
<td>0x00-0xFE (0-254), even values only</td>
<td>128</td>
</tr>
<tr>
<td>[7:2]</td>
<td>0x00-0xFC (0-252), in steps of 4</td>
<td>64</td>
</tr>
<tr>
<td>[7:3]</td>
<td>0x00-0xFB (0-248), in steps of 8</td>
<td>32</td>
</tr>
<tr>
<td>[7:4]</td>
<td>0x00-0xF0 (0-240), in steps of 16</td>
<td>16</td>
</tr>
</tbody>
</table>
Unimplemented priority bits are RAZ/WI.

This field resets to 0.

## Accessing the ICV_PMR

Accesses to this register use the following encodings:

\[
MRC\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opcl}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && HSTR_EL2.TI2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.TI2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_PMR;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IM0 == '1' then
    return ICV_PMR;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    return ICV_PMR;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IM0 == '1' then
    return ICV_PMR;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
  elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
      UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
      AArch32.TakeMonitorTrapException();
    else
      return ICC_PMR;
  elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
      UNDEFINED;
    else
      return ICC_PMR;

MCR\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opcl}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_PMR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        AArch32.TakeMonitorTrapException();
else
    ICC_PMR = R[t];
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        AArch32.TakeMonitorTrapException();
else
    ICC_PMR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
else
    ICC_PMR = R[t];
ICV_RPR, Interrupt Controller Virtual Running Priority Register

The ICV_RPR characteristics are:

**Purpose**

Indicates the Running priority of the virtual CPU interface.

**Configuration**

AArch32 System register ICV_RPR performs the same function as AArch64 System register ICV_RPR_EL1.

**Attributes**

ICV_RPR is a 32-bit register.

**Field descriptions**

The ICV_RPR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td>Priority</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The current running priority on the virtual CPU interface. This is the group priority of the current active virtual interrupt.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

---

**Note**

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

---

**Accessing the ICV_RPR**

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elif PSTATE.EL == EL1 then
  if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
elif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.TC == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
    AArch32.TakeHypTrapException(0x03);
elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
    return ICV_RPR;
elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
    return ICV_RPR;
elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FMO == '1' then
    return ICV_RPR;
elif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.IMO == '1' then
    return ICV_RPR;
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
else
  return ICC_RPR;
elif PSTATE.EL == EL2 then
  if ICC_HSRE.SRE == '0' then
    UNDEFINED;
elif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
elif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
    AArch32.TakeMonitorTrapException();
else
  return ICC_RPR;
elif PSTATE.EL == EL3 then
  if ICC_MSRE.SRE == '0' then
    UNDEFINED;
else
  return ICC_RPR;
The ID_AFR0 characteristics are:

**Purpose**

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, MIDR.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_AFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_AFR0_EL1[31:0].

**Attributes**

ID_AFR0 is a 32-bit register.

**Field descriptions**

The ID_AFR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31:16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15:12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11:8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7:4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3:0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**Accessing the ID_AFR0**

Accesses to this register use the following encodings:
### ID_AFR0, Auxiliary Feature Register 0

MRC{<c>}{<q>}{<coproc>}, {#}{<opc1>}, <Rt>, <CRn>, <CRm>{, {#}{<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_AFR0;
elsif PSTATE.EL == EL2 then
  return ID_AFR0;
elsif PSTATE.EL == EL3 then
  return ID_AFR0;
ID_DFR0, Debug Feature Register 0

The ID_DFR0 characteristics are:

**Purpose**

Provides top level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, MIDR.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_DFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_DFR0_EL1[31:0].

**Attributes**

ID_DFR0 is a 32-bit register.

**Field descriptions**

The ID_DFR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| TraceFilt | PerfMon | MProfDbg | MMapTrc | CopTrc | MMapDbg | CopSDbg | CopDbg |

**TraceFilt, bits [31:28]**

Armv8.4 Self-hosted Trace Extension version. Defined values are:

<table>
<thead>
<tr>
<th>TraceFilt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Armv8.4 Self-hosted Trace Extension not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Armv8.4 Self-hosted Trace Extension implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-Trace implements the functionality added by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

**PerfMon, bits [27:24]**

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in Alternative ID scheme used for the Performance Monitors Extension version.

Defined values are:
**PerfMon**

| \(0b0000\) | Performance Monitors Extension not implemented. |
| \(0b0001\) | Performance Monitors Extension version 1 implemented, PMUv1. |
| \(0b0010\) | Performance Monitors Extension version 2 implemented, PMUv2. |
| \(0b0011\) | Performance Monitors Extension version 3 implemented, PMUv3. |
| \(0b0100\) | PMUv3 for Armv8.1. As \(0b0011\), and also includes support for:  
  * Extended 16-bit PMEVTYPE<\(n\>).evtCount field.  
  * If EL2 is implemented, the HDCR.HPMD control bit. |
| \(0b0101\) | PMUv3 for Armv8.4. As \(0b0100\) and also includes support for the PMMIR register. |
| \(0b0110\) | PMUv3 for Armv8.5. As \(0b0101\) and also includes support for:  
  * 64-bit event counters.  
  * If EL2 is implemented, the HDCR.HCCD control bit.  
  * If EL3 is implemented, the SDCR.SCCD control bit. |
| \(0b1111\) | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value in new implementations. |

ARMv8.1-PMU implements the functionality added by the value \(0b0100\).

ARMv8.4-PMU implements the functionality added by the value \(0b0101\).

ARMv8.5-PMU implements the functionality added by the value \(0b0110\).

All other values are reserved.

In any Armv8 implementation, the values \(0b0001\) and \(0b0010\) are not permitted.

From Armv8.1, the value \(0b0011\) is not permitted.

From Armv8.4, the value \(0b0100\) is not permitted.

From Armv8.5, the value \(0b0101\) is not permitted.

**Note**

In Armv7, the value \(0b0000\) can mean that PMUv1 is implemented. PMUv1 is not permitted in an Armv8 implementation.

---

**MProfDbg, bits [23:20]**

M Profile Debug. Support for memory-mapped debug model for M profile processors. Defined values are:

<table>
<thead>
<tr>
<th>MProfDbg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0000)</td>
<td>Not supported.</td>
</tr>
<tr>
<td>(0b0001)</td>
<td>Support for M profile Debug architecture, with memory-mapped access.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the only permitted value is \(0b0000\).

**MMapTrc, bits [19:16]**

Memory Mapped Trace. Support for memory-mapped trace model. Defined values are:

<table>
<thead>
<tr>
<th>MMapTrc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0000)</td>
<td>Not supported.</td>
</tr>
<tr>
<td>(0b0001)</td>
<td>Support for Arm trace architecture, with memory-mapped access.</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A, the permitted values are 0b0000 and 0b0001.

See the ETM Architecture Specification for more information.

**CopTrc, bits [15:12]**

Support for System registers-based trace model, using registers in the coproc == 0b1110 encoding space. Defined values are:

<table>
<thead>
<tr>
<th>CopTrc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Arm trace architecture, with System registers access.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

See the ETM Architecture Specification for more information.

**MMapDbg, bits [11:8]**

Memory Mapped Debug. Support for v7 memory-mapped debug model, for A and R profile processors.

In Armv8-A, this field is RES0.

The optional memory map defined by Armv8 is not compatible with Armv7.

**CopSDbg, bits [7:4]**

Support for a System registers-based Secure debug model, using registers in the coproc == 0b1110 encoding space, for an A profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

**CopDbg, bits [3:0]**

Support for System registers-based debug model, using registers in the coproc == 0b1110 encoding space, for A and R profile processors. Defined values are:

<table>
<thead>
<tr>
<th>CopDbg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for Armv6, v6 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for Armv6, v6.1 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Support for Armv7, v7 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Support for Armv7, v7.1 Debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Support for Armv8 debug architecture, with System registers access.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Support for Armv8 debug architecture, with System registers access, and Virtualization Host extensions.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Support for Armv8.2 debug architecture.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Support for Armv8.4 debug architecture.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In any Armv8 implementation, the values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted.

If ARMv8.1-VHE is not implemented, the only permitted value is 0b0110.

In an Armv8.0 implementation, the value 0b1000 is not permitted.
Accessing the ID_DFR0

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}\ <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_DFR0;
  end
elsif PSTATE.EL == EL2 then
  return ID_DFR0;
elsif PSTATE.EL == EL3 then
  return ID_DFR0;
The ID_DFR1 characteristics are:

**Purpose**

Provides top level information about the debug system in AArch32.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_DFR1 bits [31:0] are architecturally mapped to AArch64 System register ID_DFR1_EL1[31:0].

This register is present only from Armv8.6. Otherwise, direct accesses to ID_DFR1 are RES0.

**Attributes**

ID_DFR1 is a 32-bit register.

**Field descriptions**

The ID_DFR1 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>MTPMU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**MTPMU, bits [3:0]**

From Armv8.6:

Multi-threaded PMU extension. Defined values are:

<table>
<thead>
<tr>
<th>MTPMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ARMv8.6-MTPMU not implemented. If PMUV3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTPYPER&lt;n&gt;.MT are read/write or RES0.</td>
</tr>
<tr>
<td>0b0001</td>
<td>ARMv8.6-MTPMU implemented and PMEVTPYPER&lt;n&gt;.MT are read/write. When ARMv8.6-MTPMU is disabled, the Effective values of PMEVTPYPER&lt;n&gt;.MT are 0.</td>
</tr>
<tr>
<td>0b1111</td>
<td>ARMv8.6-MTPMU not implemented. If PMUV3 is implemented, PMEVTPYPER&lt;n&gt;.MT are RES0.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-MTPMU implements the functionality identified by the value 0b0001.

In an Armv8.6-compliant implementation that includes PMUV3, the value 0b0000 is not permitted.

In an implementation that does not include PMUV3, the value 0b0001 is not permitted.
Otherwise:

Reserved, RES0.

**Accessing the ID_DFR1**

Accesses to this register use the following encodings:

\[ \text{MRC\{c\}\{q\} } \text{coproc}, \{\#\text{opc1}\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{\#\text{opc2}\} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_DFR1) || boolean IMPLEMENTATION_DEFINED "ID_DFR1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_DFR1) || boolean IMPLEMENTATION_DEFINED "ID_DFR1 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_DFR1;
    endif
elsif PSTATE.EL == EL2 then
    return ID_DFR1;
else
    if PSTATE.EL == EL3 then
        return ID_DFR1;
    endif
}}
ID_ISAR0, Instruction Set Attribute Register 0

The ID_ISAR0 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR0 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR0_EL1[31:0].

**Attributes**

ID_ISAR0 is a 32-bit register.

**Field descriptions**

The ID_ISAR0 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Divide</td>
<td>Debug</td>
<td>Coproc</td>
<td>CmpBranch</td>
<td>BitField</td>
<td>BitCount</td>
<td>Swap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**Divide, bits [27:24]**

Indicates the implemented Divide instructions. Defined values are:

<table>
<thead>
<tr>
<th>Divide</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds SDIV and UDIV in the T32 instruction set.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds SDIV and UDIV in the A32 instruction set.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**Debug, bits [23:20]**

Indicates the implemented Debug instructions. Defined values are:

<table>
<thead>
<tr>
<th>Debug</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds BKPT.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.
Coproc, bits [19:16]

Indicates the implemented System register access instructions. Defined values are:

<table>
<thead>
<tr>
<th>Coproc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds generic CDP, LDC, MCR, MRC, and STC.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and adds generic MCRR and MRRC.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds generic MCRR2 and MRRC2.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>CmpBranch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds CBNZ and CBZ.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

BitField, bits [11:8]

Indicates the implemented BitField instructions. Defined values are:

<table>
<thead>
<tr>
<th>BitField</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds BFC, BFI, SBFX, and UBFX.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions. Defined values are:

<table>
<thead>
<tr>
<th>BitCount</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds CLZ.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>Swap</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds SWP and SWPB.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.
Accessing the ID_ISAR0

Accesses to this register use the following encodings:

\[ \text{MRC}\{c\}\{q\} \text{ <coproc>}, \{#\text{<opc1>}, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{#\text{<opc2>}} \]  

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\text{if } \text{PSTATE.EL == EL0 then} \\
\quad \text{UNDEFINED;}
\text{elsif PSTATE.EL == EL1 then} \\
\quad \quad \text{if EL2Enabled() } \&\& \text{ !ELUsingAArch32(EL2) } \&\& \text{ HSTR_EL2.T0 == '1' then} \\
\quad \quad \quad \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03);} \\
\quad \quad \text{elsif EL2Enabled() } \&\& \text{ ELUsingAArch32(EL2) } \&\& \text{ HSTR.T0 == '1' then} \\
\quad \quad \quad \text{AArch32.TakeHypTrapException(0x03);} \\
\quad \quad \text{elsif EL2Enabled() } \&\& \text{ !ELUsingAArch32(EL2) } \&\& \text{ HCR_EL2.TID3 == '1' then} \\
\quad \quad \quad \text{AArch64.AArch32SystemAccessTrap(EL2, 0x03);} \\
\quad \quad \text{elsif EL2Enabled() } \&\& \text{ ELUsingAArch32(EL2) } \&\& \text{ HCR.TID3 == '1' then} \\
\quad \quad \quad \text{AArch32.TakeHypTrapException(0x03);} \\
\quad \text{else} \\
\quad \quad \text{return ID_ISAR0;} \\
\text{elsif PSTATE.EL == EL2 then} \\
\quad \text{return ID ISAR0;} \\
\text{elsif PSTATE.EL == EL3 then} \\
\quad \text{return ID_ISAR0;} \\
\end{align*}
\]
The ID_ISAR1 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state. Must be interpreted with ID_ISAR0, ID_ISAR2, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR1 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR1_EL1[31:0].

**Attributes**

ID_ISAR1 is a 32-bit register.

**Field descriptions**

The ID_ISAR1 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
| Jazelle | Interwork | Immediate | IfThen | Extend | Except_AR | Except | Endian |

**Jazelle, bits [31:28]**

Indicates the implemented Jazelle extension instructions. Defined values are:

<table>
<thead>
<tr>
<th>Jazelle</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No support for Jazelle.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the BXJ instruction, and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**Interwork, bits [27:24]**

Indicates the implemented Interworking instructions. Defined values are:

<table>
<thead>
<tr>
<th>Interwork</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the BX instruction, and the T bit in the PSR.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0011.
Immediate, bits [23:20]

Indicates the implemented data-processing instructions with long immediates. Defined values are:

<table>
<thead>
<tr>
<th>Immediate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds:</td>
</tr>
<tr>
<td></td>
<td>• The MOVT instruction</td>
</tr>
<tr>
<td></td>
<td>• The MOV instruction encodings with zero-extended 16-bit immediates.</td>
</tr>
<tr>
<td></td>
<td>• The T32 ADD and SUB instruction encodings with zero-extended 12-bit</td>
</tr>
<tr>
<td></td>
<td>immediates, and the other ADD, ADR, and SUB encodings cross-referenced</td>
</tr>
<tr>
<td></td>
<td>by the pseudocode for those encodings.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>IfThen</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the IT instructions, and the IT bits in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

Extend, bits [15:12]

Indicates the implemented Extend instructions. Defined values are:

<table>
<thead>
<tr>
<th>Extend</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No scalar sign-extend or zero-extend instructions are implemented,</td>
</tr>
<tr>
<td></td>
<td>where scalar instructions means non-Advanced SIMD instructions.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SXTB, SXTH, UXTB, and UXTH instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16,</td>
</tr>
<tr>
<td></td>
<td>SXTAH, UXTB16, UXTAB, UXTAB16, and UXTAH instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

Except_AR, bits [11:8]

Indicates the implemented A and R profile exception-handling instructions. Defined values are:

<table>
<thead>
<tr>
<th>Except_AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SRS and RFE instructions, and the A and R profile forms of the</td>
</tr>
<tr>
<td></td>
<td>CPS instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set. Defined values are:
Except | Meaning
--- | ---
0b0000 | Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented.
0b0001 | Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions.

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**Endian, bits [3:0]**

Indicates the implemented Endian instructions. Defined values are:

<table>
<thead>
<tr>
<th>Endian</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SETEND instruction, and the E bit in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

**Accessing the ID_ISAR1**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_ISAR1;
elsif PSTATE.EL == EL2 then
  return ID_ISAR1;
elsif PSTATE.EL == EL3 then
  return ID_ISAR1;
ID_ISAR2, Instruction Set Attribute Register 2

The ID_ISAR2 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR3, ID_ISAR4, and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR2 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR2_EL1[31:0].

**Attributes**

ID_ISAR2 is a 32-bit register.

**Field descriptions**

The ID_ISAR2 bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Reversal | PSR_AR | MultU | MultS | Mult | MultiAccessInt | MemHint | LoadStore |

**Reversal, bits [31:28]**

Indicates the implemented Reversal instructions. Defined values are:

<table>
<thead>
<tr>
<th>Reversal</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the REV, REV16, and REVSH instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the RBIT instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**PSR_AR, bits [27:24]**

Indicates the implemented A and R profile instructions to manipulate the PSR. Defined values are:

<table>
<thead>
<tr>
<th>PSR_AR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the SUBS PC,LR,#N instruction.
MultU, bits [23:20]

Indicates the implemented advanced unsigned Multiply instructions. Defined values are:

<table>
<thead>
<tr>
<th>MultU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the UMULL and UMLAL instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the UMAAL instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

MultS, bits [19:16]

Indicates the implemented advanced signed Multiply instructions. Defined values are:

<table>
<thead>
<tr>
<th>MultS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SMULL and SMLAL instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMLUBB, SMLULT, SMLUTB, SMLUTT, SMLUWB, and SMLUWT instructions. Also adds the Q bit in the PSRs.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and adds the SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSDLX, SMMUL, SMMULR, SMMULM, SMMULM, SMUSD, and SMUSDX instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0011.

Mult, bits [15:12]

Indicates the implemented additional Multiply instructions. Defined values are:

<table>
<thead>
<tr>
<th>Mult</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No additional instructions implemented. This means only MUL is implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the MLA instruction.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the MLS instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

MultiAccessInt, bits [11:8]

Indicates the support for interruptible multi-access instructions. Defined values are:

<table>
<thead>
<tr>
<th>MultiAccessInt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No support. This means the LDM and STM instructions are not interruptible.</td>
</tr>
<tr>
<td>0b0001</td>
<td>LDM and STM instructions are restartable.</td>
</tr>
<tr>
<td>0b0010</td>
<td>LDM and STM instructions are continuable.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

MemHint, bits [7:4]

Indicates the implemented Memory Hint instructions. Defined values are:
MemHint | Meaning
--- | ---
0b0000 | None implemented.
0b0001 | Adds the PLD instruction.
0b0010 | Adds the PLD instruction. (0b0001 and 0b0010 have identical effects.)
0b0011 | As for 0b0001 (or 0b0010), and adds the PLI instruction.
0b0100 | As for 0b0011, and adds the PLDW instruction.

All other values are reserved.

In Armv8-A the only permitted value is 0b0100.

### LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions. Defined values are:

<table>
<thead>
<tr>
<th>LoadStore</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No additional load/store instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the LDRD and STRD instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds the PLDW instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

### Accessing the ID_ISAR2

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>\{, \{#\}<opc2>\}}
\]

<table>
<thead>
<tr>
<th>proc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR2;
    elsif PSTATE.EL == EL2 then
        return ID_ISAR2;
    elsif PSTATE.EL == EL3 then
        return ID_ISAR2;

ID_ISAR3, Instruction Set Attribute Register 3

The ID_ISAR3 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR4](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR3 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR3_EL1][31:0].

**Attributes**

ID_ISAR3 is a 32-bit register.

**Field descriptions**

The ID_ISAR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| T32EE | TrueNOP | T32Copy | TabBranch | SynchPrim | SVC | SIMD | Saturate |

**T32EE, bits [31:28]**

Indicates the implemented T32EE instructions. Defined values are:

<table>
<thead>
<tr>
<th>T32EE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**TrueNOP, bits [27:24]**

Indicates the implemented true NOP instructions. Defined values are:

<table>
<thead>
<tr>
<th>TrueNOP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. This means there are no NOP instructions that do not have any register dependencies.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.
**T32Copy, bits [23:20]**

Indicates the support for T32 non flag-setting MOV instructions. Defined values are:

<table>
<thead>
<tr>
<th>T32Copy</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported. This means that in the T32 instruction set, encoding T1 of the MOV (register) instruction does not support a copy from a low register to a low register.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds support for T32 instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**TabBranch, bits [19:16]**

Indicates the implemented Table Branch instructions in the T32 instruction set. Defined values are:

<table>
<thead>
<tr>
<th>TabBranch</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the TBB and TBH instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**SynchPrim, bits [15:12]**

Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions. Defined values are:

<table>
<thead>
<tr>
<th>SynchPrim</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If SynchPrim_frac == 0b000, no Synchronization Primitives implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>If SynchPrim_frac == 0b000, adds the LDREX and STREX instructions. If SynchPrim_frac == 0b011, also adds the CLREX, LDREXB, STREXB, and STREXH instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>If SynchPrim_frac == 0b000, as for [0b001, 0b011] and also adds the LDREXD and STREXD instructions.</td>
</tr>
</tbody>
</table>

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A the only permitted value is 0b0010.

**SVC, bits [11:8]**

Indicates the implemented SVC instructions. Defined values are:

<table>
<thead>
<tr>
<th>SVC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SVC instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**SIMD, bits [7:4]**

Indicates the implemented SIMD instructions. Defined values are:
### SIMD

<table>
<thead>
<tr>
<th>SIMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SSAT and USAT instructions, and the Q bit in the PSRs.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0001, and adds the PKHTB, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0011.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports floating-point and Advanced SIMD instructions, MVFR0, MVFR1, and MVFR2 give information about the implemented Advanced SIMD instructions.

### Saturate, bits [3:0]

Indicates the implemented Saturate instructions. Defined values are:

<table>
<thead>
<tr>
<th>Saturate</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. This means no non-Advanced SIMD saturate instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

### Accessing the ID_ISAR3

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_ISAR3;
  endif
elsif PSTATE.EL == EL2 then
  return ID_ISAR3;
elsif PSTATE.EL == EL3 then
  return ID_ISAR3;
```
ID_ISAR4, Instruction Set Attribute Register 4

The ID_ISAR4 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with `ID_ISAR0`, `ID_ISAR1`, `ID_ISAR2`, `ID_ISAR3`, and `ID_ISAR5`.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR4 bits [31:0] are architecturally mapped to AArch64 System register `ID_ISAR4_EL1[31:0]`.

**Attributes**

ID_ISAR4 is a 32-bit register.

**Field descriptions**

The ID_ISAR4 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SWP_frac</strong></td>
<td><strong>PSR_M</strong></td>
<td><strong>SynchPrim_frac</strong></td>
<td><strong>Barrier</strong></td>
<td><strong>SMC</strong></td>
<td><strong>Writeback</strong></td>
<td><strong>WithShifts</strong></td>
<td><strong>Unpriv</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SWP_frac, bits [31:28]**

Indicates support for the memory system locking the bus for SWP or SWPB instructions. Defined values are:

<table>
<thead>
<tr>
<th>SWP_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SWP or SWPB instructions not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SWP or SWPB implemented but only in a uniprocessor context. SWP and SWPB do not guarantee whether memory accesses from other masters can come between the load memory access and the store memory access of the SWP or SWPB.</td>
</tr>
</tbody>
</table>

All other values are reserved. This field is valid only if the `ID_ISAR0`.Swap_instrs field is 0b0000.

In Armv8-A the only permitted value is 0b0000.

**PSR_M, bits [27:24]**

Indicates the implemented M profile instructions to modify the PSRs. Defined values are:

<table>
<thead>
<tr>
<th>PSR_M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the M profile forms of the CPS, MRS, and MSR instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.
**SynchPrim_frac, bits [23:20]**

Used in conjunction with ID_ISAR3.SynchPrim to indicate the implemented Synchronization Primitive instructions. Possible values are:

<table>
<thead>
<tr>
<th>SynchPrim_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If SynchPrim == 0b0000, no Synchronization Primitives implemented. If SynchPrim == 0b0001, adds the LDREX and STREX instructions. If SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB, STREXH, LDREXD, and STREXD instructions.</td>
</tr>
<tr>
<td>0b0011</td>
<td>If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions.</td>
</tr>
</tbody>
</table>

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A the only permitted value is 0b0000.

**Barrier, bits [19:16]**

Indicates the implemented Barrier instructions in the A32 and T32 instruction sets. Defined values are:

<table>
<thead>
<tr>
<th>Barrier</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. Barrier operations are provided only as System instructions in the (coproc==0b1111) encoding space.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the DMB, DSB, and ISB barrier instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**SMC, bits [15:12]**

Indicates the implemented SMC instructions. Defined values are:

<table>
<thead>
<tr>
<th>SMC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the SMC instruction.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**Writeback, bits [11:8]**

Indicates the support for Writeback addressing modes. Defined values are:

<table>
<thead>
<tr>
<th>Writeback</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions support writeback addressing modes. These instructions support all of their writeback addressing modes.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds support for all of the writeback addressing modes.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

**WithShifts, bits [7:4]**

Indicates the support for instructions with shifts. Defined values are:
All other values are reserved.

In Armv8-A the only permitted value is 0b0100.

### Unpriv, bits [3:0]

Indicates the implemented unprivileged instructions. Defined values are:

<table>
<thead>
<tr>
<th>Unpriv</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None implemented. No T variant instructions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Adds the LDRBT, LDRT, STRBT, and STRT instructions.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

### Accessing the ID_ISAR4

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```ruby
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_ISAR4;
  elsif PSTATE.EL == EL2 then
    return ID_ISAR4;
elsif PSTATE.EL == EL3 then
  return ID_ISAR4;
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ID_ISAR5, Instruction Set Attribute Register 5

The ID_ISAR5 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, and ID_ISAR4.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR5 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR5_EL1[31:0].

**Attributes**

ID_ISAR5 is a 32-bit register.

**Field descriptions**

The ID_ISAR5 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VCMA | RDM | RES0 | CRC32 | SHA2 | SHA1 | AES | SEVL |

**VCMA, bits [31:28]**

From Armv8.3:

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors. Defined values are:

<table>
<thead>
<tr>
<th>VCMA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The VCMLA and VCADD instructions are not implemented in AArch32.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The VCMLA and VCADD instructions are implemented in AArch32.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0, Armv8.1 and Armv8.2 the only permitted value is 0b0000.

From Armv8.3 the only permitted value is 0b0001. This feature is identified as ARMv8.3-CompNum.

Otherwise:

Reserved, RES0.

**RDM, bits [27:24]**

From Armv8.1:

Indicates whether the VQRDMLAH and VQRDMLSH instructions are implemented in AArch32 state. Defined values are:
<table>
<thead>
<tr>
<th>RDM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No VQRDMLAH and VQRDMLSH instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VQRDMLAH and VQRDMLSH instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-RDMA implements the functionality identified by the value 0b0001.

From Armv8.1 the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

Bits [23:20]

Reserved, RES0.

CRC32, bits [19:16]

Indicates whether the CRC32 instructions are implemented in AArch32 state.

<table>
<thead>
<tr>
<th>CRC32</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No CRC32 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0 the permitted values are 0b0000 and 0b0001.

From Armv8.1 the only permitted value is 0b0001.

SHA2, bits [15:12]

Indicates whether the SHA2 instructions are implemented in AArch32 state.

<table>
<thead>
<tr>
<th>SHA2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA2 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

SHA1, bits [11:8]

Indicates whether the SHA1 instructions are implemented in AArch32 state.

<table>
<thead>
<tr>
<th>SHA1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No SHA1 instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

AES, bits [7:4]

Indicates whether the AES instructions are implemented in AArch32 state.
### AES

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No AES instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AESE, AESD, AESMC, and AESIMC implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, plus VMULL (polynomial) instructions operating on 64-bit data quantities.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

### SEVL, bits [3:0]

Indicates whether the SEVL instruction is implemented in AArch32 state.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SEVL is implemented as a NOP.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SEVL is implemented as Send Event Local.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

### Accessing the ID_ISAR5

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td></td>
<td></td>
<td>0b101</td>
</tr>
</tbody>
</table>

```latex
\begin{verbatim}
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_ISAR5;
  end
elsif PSTATE.EL == EL2 then
  return ID_ISAR5;
elsif PSTATE.EL == EL3 then
  return ID_ISAR5;
end
\end{verbatim}
```
ID_ISAR6, Instruction Set Attribute Register 6

The ID_ISAR6 characteristics are:

**Purpose**

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with ID_ISAR0, ID_ISAR1, ID_ISAR2, ID_ISAR3, ID_ISAR4 and ID_ISAR5.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_ISAR6 bits [31:0] are architecturally mapped to AArch64 System register ID_ISAR6_EL1[31:0].

This register is present only from Armv8.2. Otherwise, direct accesses to ID_ISAR6 are RES0 from EL1, EL2, and EL3.

**Attributes**

ID_ISAR6 is a 32-bit register.

**Field descriptions**

The ID_ISAR6 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | I8MM | BF16 | SPECRES | SB | FHM | DP | JSCVT |

**Bits [31:28]**

Reserved, RES0.

**I8MM, bits [27:24]**

**When ARMv8.2-AA32I8MM is implemented:**

Indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>I8MM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Int8 matrix multiplication instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VSMMMLA, VSUDOT, VUMMLA, VUSMMLA, and VUSDOT instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-AA32I8MM implements the functionality identified by 0b0001.

**Otherwise:**

Reserved, RES0.
BF16, bits [23:20]

When ARMv8.2-AA32BF16 is implemented:

Indicates support for Advanced SIMD and floating-point BF16 instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>BF16</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>BF16 instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VCVT, VCVTB, VCVTT, VDOT, VFMA, and VMMAL instructions with BF16 operand or result types are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-AA32BF16 implements the functionality identified by 0b0001.

Otherwise:

Reserved, RES0.

SPECRES, bits [19:16]

Indicates support for Speculation invalidation instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>SPECRES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>CFPRCTX, DVPRCTX, and CPPRCTX instructions are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [15:12]

Indicates support for SB instruction in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>SB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SB instruction is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SB instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

FHM, bits [11:8]

From Armv8.2:

Indicates support for Advanced SIMD and floating-point VFMA and VFMSL instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>FHM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>VFMA and VFMSL instructions not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VFMA and VFMSL instructions implemented.</td>
</tr>
</tbody>
</table>

ARMv8.2-FHM implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

DP, bits [7:4]
From Armv8.2:

Indicates support for dot product instructions in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>DP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No dot product instructions implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>VUDOT and VSDOT instructions implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-DotProd implements the functionality identified by the value 0b0001.

Otherwise:

Reserved, RES0.

JSCVT, bits [3:0]

From Armv8.3:

Indicates support for the Javascript conversion instruction in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>JSCVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The VJCVT instruction is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The VJCVT instruction is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0, Armv8.1 and Armv8.2 the only permitted value is 0b0000.

From Armv8.3 the only permitted value is 0b0001. This feature is identified as ARMv8.3.JSConv.

Otherwise:

Reserved, RES0.

**Accessing the ID_ISAR6**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_ISAR6) || boolean IMPLEMENTATION_DEFINED "ID_ISAR6 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_ISAR6) || boolean IMPLEMENTATION_DEFINED "ID_ISAR6 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR6;
    endif
elsif PSTATE.EL == EL2 then
    return ID_ISAR6;
elsif PSTATE.EL == EL3 then
    return ID_ISAR6;
endif
The ID_MMFR0 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR1, ID_MMFR2, ID_MMFR3, and ID_MMFR4.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_MMFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR0_EL1[31:0].

**Attributes**

ID_MMFR0 is a 32-bit register.

**Field descriptions**

The ID_MMFR0 bit assignments are:

<table>
<thead>
<tr>
<th>InnerShr</th>
<th>FCSE</th>
<th>AuxReg</th>
<th>TCM</th>
<th>ShareLvl</th>
<th>OuterShr</th>
<th>PMSA</th>
<th>VMSA</th>
</tr>
</thead>
</table>

**InnerShr, bits [31:28]**

Innermost Shareability. Indicates the innermost shareability domain implemented. Defined values are:

<table>
<thead>
<tr>
<th>InnerShr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Implemented as Non-cacheable.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented with hardware coherency support.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Shareability ignored.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0.ShareLvl having the value 0b0001.

When ID_MMFR0.ShareLvl is zero, this field is UNK.

**FCSE, bits [27:24]**

Indicates whether the implementation includes the FCSE. Defined values are:

<table>
<thead>
<tr>
<th>FCSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for FCSE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.
AuxReg, bits [23:20]

Auxiliary Registers. Indicates support for Auxiliary registers. Defined values are:

<table>
<thead>
<tr>
<th>AuxReg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Auxiliary Control Register only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for Auxiliary Fault Status Registers (AIFSR and ADFSР) and Auxiliary Control Register.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0010.

Note

Accesses to unimplemented Auxiliary registers are UNDEFINED.

TCM, bits [19:16]

Indicates support for TCMs and associated DMAs. Defined values are:

<table>
<thead>
<tr>
<th>TCM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support is IMPLEMENTATION DEFINED. Armv7 requires this setting.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for TCM only. Armv6 implementation.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for TCM and DMA, Armv6 implementation.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

ShareLvl, bits [15:12]

Shareability Levels. Indicates the number of shareability levels implemented. Defined values are:

<table>
<thead>
<tr>
<th>ShareLvl</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>One level of shareability implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Two levels of shareability implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0001.

OuterShr, bits [11:8]

Outermost Shareability. Indicates the outermost shareability domain implemented. Defined values are:

<table>
<thead>
<tr>
<th>OuterShr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Implemented as Non-cacheable.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented with hardware coherency support.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Shareability ignored.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

PMSA, bits [7:4]

Indicates support for a PMSA. Defined values are:
### PMSA

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for IMPLEMENTATION DEFINED PMSA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for PMSAv6, with a Cache Type Register implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for PMSAv7, with support for memory subsections. Armv7-R profile.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

### VMSA, bits [3:0]

Indicates support for a VMSA. Defined values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for IMPLEMENTATION DEFINED VMSA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Support for VMSAv6, with Cache and TLB Type Registers implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Support for VMSAv7, with support for remapping and the Access flag. Armv7-A profile.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As for 0b0100, and adds support for the Long-descriptor translation table format.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0101.

### Accessing the ID_MMFR0

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & !ELUsingAArch32(EL2) & HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & ELUsingAArch32(EL2) & HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() & !ELUsingAArch32(EL2) & HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & ELUsingAArch32(EL2) & HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
else
  return ID_MMFR0;
elsif PSTATE.EL == EL2 then
  return ID_MMFR0;
elsif PSTATE.EL == EL3 then
  return ID_MMFR0;
The ID_MMFR1 characteristics are:

### Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0, ID_MMFR2, ID_MMFR3, and ID_MMFR4.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

### Configuration

AArch32 System register ID_MMFR1 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR1_EL1[31:0].

### Attributes

ID_MMFR1 is a 32-bit register.

### Field descriptions

The ID_MMFR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-28</td>
<td><strong>BPred</strong></td>
</tr>
<tr>
<td>27-24</td>
<td><strong>L1TstCln</strong></td>
</tr>
<tr>
<td>23-20</td>
<td><strong>L1Uni</strong></td>
</tr>
<tr>
<td>19-16</td>
<td><strong>L1Hvd</strong></td>
</tr>
<tr>
<td>15-12</td>
<td><strong>L1UniSW</strong></td>
</tr>
<tr>
<td>11-8</td>
<td><strong>L1HvdSW</strong></td>
</tr>
<tr>
<td>7-4</td>
<td><strong>L1UniVA</strong></td>
</tr>
<tr>
<td>3-0</td>
<td><strong>L1HvdVA</strong></td>
</tr>
</tbody>
</table>

#### BPred, bits [31:28]

Branch Predictor. Indicates branch predictor management requirements. Defined values are:

<table>
<thead>
<tr>
<th>BPred</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No branch predictor, or no MMU present. Implies a fixed MPU configuration.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Branch predictor requires flushing on:</td>
</tr>
<tr>
<td></td>
<td>• Enabling or disabling a stage of address translation.</td>
</tr>
<tr>
<td></td>
<td>• Writing new data to instruction locations.</td>
</tr>
<tr>
<td></td>
<td>• Writing new mappings to the translation tables.</td>
</tr>
<tr>
<td></td>
<td>• Changes to the TTBR0, TTBR1, or TTBCR registers.</td>
</tr>
<tr>
<td></td>
<td>• Changes to the ContextID or ASID, or to the FCSE ProcessID if this is supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Branch predictor requires flushing on:</td>
</tr>
<tr>
<td></td>
<td>• Enabling or disabling a stage of address translation.</td>
</tr>
<tr>
<td></td>
<td>• Writing new data to instruction locations.</td>
</tr>
<tr>
<td></td>
<td>• Writing new mappings to the translation tables.</td>
</tr>
<tr>
<td></td>
<td>• Any change to the TTBR0, TTBR1, or TTBCR registers without a change to the corresponding ContextID or ASID, or FCSE ProcessID if this is supported.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Branch predictor requires flushing only on writing new data to instruction locations.</td>
</tr>
<tr>
<td>0b0100</td>
<td>For execution correctness, branch predictor requires no flushing at any time.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0010, 0b0011, or 0b0100. For values other than 0b0000 and 0b0100 the Arm Architecture Reference Manual, or the product documentation, might give more information about the required maintenance.
**L1TstCln, bits [27:24]**

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations. Defined values are:

<table>
<thead>
<tr>
<th>L1TstCln</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 data cache test and clean operations are:</td>
</tr>
<tr>
<td></td>
<td>• Test and clean data cache.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Test, clean, and invalidate data cache.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**L1Uni, bits [23:20]**

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1Uni</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported entire Level 1 cache operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate cache, including branch predictor if appropriate.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate branch predictor, if appropriate.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Clean cache, using a recursive model that uses the cache dirty status bit.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate cache, using a recursive model that uses the cache dirty status bit.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**L1Hvd, bits [19:16]**

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1Hvd</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported entire Level 1 cache operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction cache, including branch predictor if appropriate.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate branch predictor, if appropriate.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache and instruction cache, including branch predictor if appropriate.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0010, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache, using a recursive model that uses the cache dirty status bit.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**L1UniSW, bits [15:12]**

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation. Defined values are:
L1UniSW, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1UniVA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 unified cache line maintenance operations by VA are:</td>
</tr>
<tr>
<td></td>
<td>• Clean cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate cache line by VA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0010, and adds:</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A the only permitted value is 0b0000.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1HvdVA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache line maintenance operations by VA are:</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache line by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache line by VA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0010, and adds:</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A the only permitted value is 0b0000.
Meaning

0b0000  None supported.
0b0001  Supported Level 1 Harvard cache line maintenance operations by VA are:
  • Clean data cache line by VA.
  • Invalidate data cache line by VA.
  • Clean and invalidate data cache line by VA.
  • Clean instruction cache line by VA.
0b0010  As for 0001, and adds:
  • Invalidate branch predictor by VA, if branch predictor is implemented.

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

**Accessing the ID_MMFR1**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR1;
    end
elsif PSTATE.EL == EL2 then
    return ID_MMFR1;
elsif PSTATE.EL == EL3 then
    return ID_MMFR1;
ID_MMFR2, Memory Model Feature Register 2

The ID_MMFR2 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0, ID_MMFR1, ID_MMFR3, and ID_MMFR4.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_MMFR2 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR2_EL1[31:0].

**Attributes**

ID_MMFR2 is a 32-bit register.

**Field descriptions**

The ID_MMFR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:28</td>
<td>HWAccFlg</td>
</tr>
<tr>
<td>27:24</td>
<td>WFIStall</td>
</tr>
<tr>
<td>23:20</td>
<td>MemBarr</td>
</tr>
<tr>
<td>19:16</td>
<td>UniTLB</td>
</tr>
<tr>
<td>15:12</td>
<td>HvdTLB</td>
</tr>
<tr>
<td>11:8</td>
<td>L1HvdRng</td>
</tr>
<tr>
<td>7:4</td>
<td>L1HvdBG</td>
</tr>
<tr>
<td>3:0</td>
<td>L1HvdFG</td>
</tr>
</tbody>
</table>

**HWAccFlg, bits [31:28]**

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation. Defined values are:

<table>
<thead>
<tr>
<th>HWAccFlg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for VMSAv7 Access flag, updated in hardware.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b000.

**WFIStall, bits [27:24]**

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling. Defined values are:

<table>
<thead>
<tr>
<th>WFIStall</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for WFI stalling.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the permitted values are 0b000 and 0b001.

**MemBarr, bits [23:20]**

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc==1111) encoding space:
MemBarr

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported memory barrier System instructions are:</td>
</tr>
<tr>
<td></td>
<td>• Data Synchronization Barrier (DSB).</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Instruction Synchronization Barrier (ISB).</td>
</tr>
<tr>
<td></td>
<td>• Data Memory Barrier (DMB).</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b010.

Arm deprecates the use of these operations. ID_ISAR4.Barrier_inhrs indicates the level of support for the preferred barrier instructions.

UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation. Defined values are:

<table>
<thead>
<tr>
<th>UniTLB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported unified TLB maintenance operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate all entries in the TLB.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate TLB entry by VA.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate TLB entries by ASID match.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b010, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation</td>
</tr>
<tr>
<td>0b0100</td>
<td>As for 0b011, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate Hyp mode unified TLB entry by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate entire Non-secure PL1&amp;0 unified TLB.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate entire Hyp mode unified TLB.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As for 0b100, and adds the following operations: TLBIMVALIS, TLBIMVAALIS, TLBIMVALHIS, TLBIMVAL, TLBIMVAAL, TLBIMVALH.</td>
</tr>
<tr>
<td>0b0110</td>
<td>As for 0b101, and adds the following operations: TLBIIPAS2IS, TLBIIPAS2LIS, TLBIIPAS2, TLBIIPAS2L.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b110.

HvdTLB, bits [15:12]

If the Unified TLB field (UniTLB, bits [19:16]) is not 0000, then the meaning of this field is implementation defined. Arm deprecates the use of this field by software.

L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation. Defined values are:

<table>
<thead>
<tr>
<th>L1HvdRng</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache maintenance range operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache range by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.
L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations. Defined values are:

<table>
<thead>
<tr>
<th>L1HvdBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache background fetch operations are:</td>
</tr>
<tr>
<td></td>
<td>• Fetch instruction cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Fetch data cache range by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations. Defined values are:

<table>
<thead>
<tr>
<th>L1HvdFG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported Level 1 Harvard cache foreground fetch operations are:</td>
</tr>
<tr>
<td></td>
<td>• Fetch instruction cache range by VA.</td>
</tr>
<tr>
<td></td>
<td>• Fetch data cache range by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

Accessing the ID_MMFR2

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} <\text{coproc}, \{#<opc1>, <Rt>, <CRn>, <CRm}\{, \{#<opc2>\}}
\]

### Table

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_MMFR2;
elsif PSTATE.EL == EL2 then
  return ID_MMFR2;
elsif PSTATE.EL == EL3 then
  return ID_MMFR2;
ID_MMFR3, Memory Model Feature Register 3

The ID_MMFR3 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with ID_MMFR0, ID_MMFR1, ID_MMFR2, and ID_MMFR4.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_MMFR3 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR3_EL1[31:0].

**Attributes**

ID_MMFR3 is a 32-bit register.

**Field descriptions**

The ID_MMFR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Supersec | CMemSz | CohWalk | PAN | MaintBcst | BPMaint | CMaintSW | CMaintVA |

**Supersec, bits [31:28]**

Supersections. On a VMSA implementation, indicates whether Supersections are supported. Defined values are:

<table>
<thead>
<tr>
<th>Supersec</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Supersections supported.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Supersections not supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b1111.

**CMemSz, bits [27:24]**

Cached Memory Size. Indicates the physical memory size supported by the caches. Defined values are:

<table>
<thead>
<tr>
<th>CMemSz</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>4GB, corresponding to a 32-bit physical address range.</td>
</tr>
<tr>
<td>0b0001</td>
<td>64GB, corresponding to a 36-bit physical address range.</td>
</tr>
<tr>
<td>0b0010</td>
<td>1TB or more, corresponding to a 40-bit or larger physical address range.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000, 0b0001, and 0b0010.
CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification. Defined values are:

<table>
<thead>
<tr>
<th>CohWalk</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Updates to the translation tables require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Updates to the translation tables do not require a clean to the Point of Unification to ensure visibility by subsequent translation table walks.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

PAN, bits [19:16]

From Armv8.1:

Privileged Access Never. Indicates support for the PAN bit in CPSR, SPSR, and DSPSR in AArch32 state. Defined values are:

<table>
<thead>
<tr>
<th>PAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PAN not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PAN supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>PAN supported and ATS1CPRP and ATS1CPWP instructions supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.1-PAN implements the functionality identified by the value 0b0001.

ARMv8.2-ATS1E1 implements the functionality added by the value 0b0010.

In Armv8.1 the value 0b0000 is not permitted.

From Armv8.2, the only permitted value is 0b0010.

Otherwise:

Reserved, RES0.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast. Defined values are:

<table>
<thead>
<tr>
<th>MaintBcst</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Cache, TLB, and branch predictor operations only affect local structures.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.
**BPMAint, bits [11:8]**

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations. Defined values are:

<table>
<thead>
<tr>
<th>BPMAint</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported branch predictor maintenance operations are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate all branch predictors.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0001, and adds:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate branch predictors by VA.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0010.

**CMaintSW, bits [7:4]**

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches. Defined values are:

<table>
<thead>
<tr>
<th>CMaintSW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported hierarchical cache maintenance instructions by set/way are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache by set/way.</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache by set/way.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache by set/way.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

**CMaintVA, bits [3:0]**

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches. Defined values are:

<table>
<thead>
<tr>
<th>CMaintVA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported hierarchical cache maintenance operations by VA are:</td>
</tr>
<tr>
<td></td>
<td>• Invalidate data cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean data cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Clean and invalidate data cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate instruction cache by VA.</td>
</tr>
<tr>
<td></td>
<td>• Invalidate all instruction cache entries.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

**Accessing the ID_MMFR3**

Accesses to this register use the following encodings:

MRC{{<c>}{<q>}}<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>proc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR3;
    endif
elsif PSTATE.EL == EL2 then
    return ID_MMFR3;
elsif PSTATE.EL == EL3 then
    return ID_MMFR3;
ID_MMFR4, Memory Model Feature Register 4

The ID_MMFR4 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

Must be interpreted with [ID_MMFR0](#), [ID_MMFR1](#), [ID_MMFR2](#), and [ID_MMFR3](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_MMFR4 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR4_EL1](#)[31:0].

**Attributes**

ID_MMFR4 is a 32-bit register.

**Field descriptions**

The ID_MMFR4 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| EVT | CCIDX | LSM | HPDS | CnP | XNX | AC2 | SpecSEI |

**EVT, bits [31:28]**

**When ARMv8.2-EVT is implemented:**

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the [HCR2](#).{TTLBIS, TOCU, TICAB, TID4} traps. Defined values are:

<table>
<thead>
<tr>
<th>EVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td><a href="#">HCR2</a>.{TTLBIS, TOCU, TICAB, TID4} traps are not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td><a href="#">HCR2</a>.{TOCU, TICAB, TID4} traps are supported.</td>
</tr>
<tr>
<td></td>
<td><a href="#">HCR2</a>.TTLBIS trap is not supported</td>
</tr>
<tr>
<td>0b0010</td>
<td><a href="#">HCR2</a>.{TTLBIS, TOCU, TICAB, TID4} traps are supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0, the only permitted value is 0b0000.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0010 when EL2 is implemented. This feature is identified as ARMv8.2-EVT.

**Otherwise:**

Reserved, RES0.

**CCIDX, bits [27:24]**
**From Armv8.3:**

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated. Defined values are:

<table>
<thead>
<tr>
<th>CCIDX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>32-bit format implemented for all levels of the CCSIDR, and the</td>
</tr>
<tr>
<td></td>
<td>CCSIDR2 register is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>64-bit format implemented for all levels of the CCSIDR, and the</td>
</tr>
<tr>
<td></td>
<td>CCSIDR2 register is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.3, the permitted values are 0b0000 and 0b0001. This feature is identified as ARMv8.3-CCIDX.

**Otherwise:**

Reserved, RAZ.

**LSM, bits [23:20]**

**From Armv8.2:**

Indicates support for LSMAOE and nTLSMD bits in HSCTRL and SCTLR. Defined values are:

<table>
<thead>
<tr>
<th>LSM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>LSMAOE and nTLSMD bits not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>LSMAOE and nTLSMD bits supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-LSMAOC implements the functionality identified by the value 0b0001.

**Otherwise:**

Reserved, RAZ.

**HPDS, bits [19:16]**

**From Armv8.2:**

Hierarchical permission disables bits in translation tables. Defined values are:

<table>
<thead>
<tr>
<th>HPDS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Disabling of hierarchical controls not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supports disabling of hierarchical controls using the TTBCR2, HPD0, TTBCR2 HPD1, and HTCR HPD bits.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED USE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-AA32HPD implements the functionality identified by the value 0b0001.

ARMv8.2-TTPBHA implements the functionality added by the value 0b0010.

**Note**

The value 0b0000 implies that the encoding for TTBCR2 is UNDEFINED.

---

ID_MMFR4, Memory Model Feature Register 4
Otherwise:

Reserved, RAZ.

**CnP, bits [15:12]**

From Armv8.2:

Common not Private translations. Defined values are:

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Common not Private translations not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Common not Private translations supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2 the only permitted value is 0b0001.

Otherwise:

Reserved, RAZ.

**XNX, bits [11:8]**

From Armv8.2:

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

<table>
<thead>
<tr>
<th>XNX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Distinction between EL0 and EL1 execute-never control at stage 2 not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Distinction between EL0 and EL1 execute-never control at stage 2 supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-TTS2UXN implements the functionality identified by the value 0b0001.

When ARMv8.2-TTS2UXN is implemented:

- If all of the following conditions are true it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4.XNX is 0b0000 or 0b0001:
  - ID_AA64MMFR1_EL1.XNX ==1.
  - EL2 cannot use AArch32.
  - EL1 can use AArch32.
- If EL2 can use AArch32 then the only permitted value is 0b0001.

Otherwise:

Reserved, RAZ.

**AC2, bits [7:4]**

Indicates the extension of the ACTLR and HACTLR registers using ACTLR2 and HACTLR2. Defined values are:

<table>
<thead>
<tr>
<th>AC2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ACTLR2 and HACTLR2 are not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>ACTLR2 and HACTLR2 are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are 0b0000 and 0b0001.
From Armv8.2, the only permitted value is 0b0001.

**SpecSEI, bits [3:0]**

**When RAS is implemented:**

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

<table>
<thead>
<tr>
<th>SpecSEI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The PE never generates an SError interrupt due to an External abort on a speculative read.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The PE might generate an SError interrupt due to an External abort on a speculative read.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Otherwise:**

Reserved, RES0. This provides no information about whether the PE generates a speculative SError interrupt.

**Accessing the ID_MMFR4**

Accesses to this register use the following encodings:

MRC{<c>}{<q>}, <coproc>, {{#}<opc1>}, <Rt>, <CRn>, <CRm>{{#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```cpp
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4) || boolean IMPLEMENTATION_DEFINED "ID_MMFR4 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4) || boolean IMPLEMENTATION_DEFINED "ID_MMFR4 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR4;
    end
elsif PSTATE.EL == EL2 then
    return ID_MMFR4;
elsif PSTATE.EL == EL3 then
    return ID_MMFR4;
else
    return ID_MMFR4;
end
```
ID_MMFR5, Memory Model Feature Register 5

The ID_MMFR5 characteristics are:

**Purpose**

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see ‘Principles of the ID scheme for fields in ID registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_MMFR5 bits [31:0] are architecturally mapped to AArch64 System register ID_MMFR5_EL1[31:0].

**Attributes**

ID_MMFR5 is a 32-bit register.

**Field descriptions**

The ID_MMFR5 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 | ETS |

**Bits [31:4]**

Reserved, RES0.

**ETS, bits [3:0]**

**From Armv8.0:**

Support for Enhanced Translation Synchronization. Defined values are:

<table>
<thead>
<tr>
<th>ETS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Enhanced Translation Synchronization is not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Enhanced Translation Synchronization is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.0-ETS implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

**Otherwise:**

Reserved, RES0.

**Accessing the ID_MMFR5**

Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR5) || boolean IMPLEMENTATION_DEFINED "ID_MMFR5 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_MMFR5) || boolean IMPLEMENTATION_DEFINED "ID_MMFR5 trapped by HCR.TID3") && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ID_MMFR5;
  elsif PSTATE.EL == EL2 then
    return ID_MMFR5;
  elsif PSTATE.EL == EL3 then
    return ID_MMFR5;
}
ID_PFR0, Processor Feature Register 0

The ID_PFR0 characteristics are:

**Purpose**

Gives top-level information about the instruction sets and other features supported by the PE in AArch32 state.

Must be interpreted with ID_PFR1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_PFR0 bits [31:0] are architecturally mapped to AArch64 System register ID_PFR0_EL1[31:0].

**Attributes**

ID_PFR0 is a 32-bit register.

**Field descriptions**

The ID_PFR0 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAS | DIT | AMU | CSV2 | State3 | State2 | State1 | State0 |

**RAS, bits [31:28]**

RAS Extension version.

<table>
<thead>
<tr>
<th>RAS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No RAS Extension.</td>
</tr>
<tr>
<td>0b0001</td>
<td>RAS Extension present.</td>
</tr>
<tr>
<td>0b0010</td>
<td>ARMv8.4-RAS present. As 0b0001, and adds support for additional ERXMISC&lt;rn&gt; System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR&lt;n&gt;STATUS and support for the optional RAS Timestamp Extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

From Armv8.4, when ARMv8.4-DFE is not implemented, and ERRIDR.NUM is zero, the permitted values are IMPLEMENTATION DEFINED 0b0001 or 0b0010. Otherwise from Armv8.4 the only permitted value is 0b0010.

ARMv8.4-RAS implements the functionality identified by the value 0b0010. In Armv8.2, the only permitted value is 0b0001. In Armv8.1 and Armv8.0, the permitted values are 0b0000 and 0b0001.

**DIT, bits [27:24]**

*From Armv8.4:*

Data Independent Timing. Defined values are:
All other values are reserved.

ARMv8.4-DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

Otherwise:

Reserved, RES0.

**AMU, bits [23:20]**

From Armv8.4:

Activity Monitors Extension. Defined values are:

<table>
<thead>
<tr>
<th>AMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Activity Monitors Extension is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AMUv1 for Armv8.4 is implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>AMUv1 for Armv8.6 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters.</td>
</tr>
</tbody>
</table>

All other values are reserved.

AMUv1 implements the functionality identified by the value 0b0001.

ARMv8.6-AMU implements the functionality identified by the value 0b0010.

Otherwise:

Reserved, RES0.

**CSV2, bits [19:16]**

From Armv8.5:

Speculative use of out of context branch targets. Defined values are:

<table>
<thead>
<tr>
<th>CSV2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>This Device does not disclose whether branch targets trained in one hardware described context can affect speculative execution in a different hardware described context.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Branch targets trained in one hardware described context can only affect speculative execution in a different hardware described context in a hard-to-determine way.</td>
</tr>
</tbody>
</table>

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

Otherwise:

Reserved, RES0.

**State3, bits [15:12]**

T32EE instruction set support. Defined values are:
### State3

<table>
<thead>
<tr>
<th>State3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>T32EE instruction set implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

### State2, bits [11:8]

Jazelle extension support. Defined values are:

<table>
<thead>
<tr>
<th>State2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Jazelle extension implemented, without clearing of JOSCR.CV on exception entry.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Jazelle extension implemented, with clearing of JOSCR.CV on exception entry.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

### State1, bits [7:4]

T32 instruction set support. Defined values are:

<table>
<thead>
<tr>
<th>State1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>T32 instruction set not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>T32 encodings before the introduction of Thumb-2 technology implemented:</td>
</tr>
<tr>
<td></td>
<td>• All instructions are 16-bit.</td>
</tr>
<tr>
<td></td>
<td>• A BL or BLX is a pair of 16-bit instructions</td>
</tr>
<tr>
<td></td>
<td>• 32-bit instructions other than BL and BLX cannot be encoded.</td>
</tr>
<tr>
<td>0b0011</td>
<td>T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0011.

### State0, bits [3:0]

A32 instruction set support. Defined values are:

<table>
<thead>
<tr>
<th>State0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>A32 instruction set not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>A32 instruction set implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0001.

### Accessing the ID_PFR0

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_PFR0;
    elsif PSTATE.EL == EL2 then
        return ID_PFR0;
elsif PSTATE.EL == EL3 then
    return ID_PFR0;
The ID_PFR1 characteristics are:

**Purpose**

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_PFR1 bits [31:0] are architecturally mapped to AArch64 System register ID_PFR1_EL1[31:0].

**Attributes**

ID_PFR1 is a 32-bit register.

**Field descriptions**

The ID_PFR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| GIC | Virt_frac | Sec_frac | GenTimer | Virtualization | MProgMod | Security | ProgMod |

**GIC, bits [31:28]**

System register GIC CPU interface. Defined values are:

<table>
<thead>
<tr>
<th>GIC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>GIC CPU interface system registers not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.</td>
</tr>
<tr>
<td>0b0011</td>
<td>System register interface to version 4.1 of the GIC CPU interface is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Virt_frac, bits [27:24]**

Virtualization fractional field. When the Virtualization field is 0b0000, determines the support for features from the ARMv7 Virtualization Extensions. Defined values are:

<table>
<thead>
<tr>
<th>Virt_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No features from the ARMv7 Virtualization Extensions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The following features of the ARMv7 Virtualization Extensions are implemented:</td>
</tr>
<tr>
<td></td>
<td>• The SCR.SIF bit, if EL3 is implemented.</td>
</tr>
<tr>
<td></td>
<td>• The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented.</td>
</tr>
<tr>
<td></td>
<td>• The MSR (banked register) and MRS (banked register) instructions.</td>
</tr>
<tr>
<td></td>
<td>• The ERET instruction.</td>
</tr>
</tbody>
</table>
All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 when EL2 is implemented.
- 0b0001 when EL2 is not implemented.

This field is only valid when the value of ID_PFR1.Virtualization is 0, otherwise it holds the value 0b0000.

---

**Note**

The ID ISAR registers do not identify whether the instructions added by the ARMv7 Virtualization Extensions are implemented.

---

### Sec_frac, bits [23:20]

Security fractional field. When the Security field is 0b0000, determines the support for features from the ARMv7 Security Extensions. Defined values are:

<table>
<thead>
<tr>
<th>Sec_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No features from the ARMv7 Security Extensions are implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The following features from the ARMv7 Security Extensions are implemented:</td>
</tr>
<tr>
<td></td>
<td>• The VBAR register.</td>
</tr>
<tr>
<td></td>
<td>• The TTBCR.PD0 and TTBCR.PD1 bits.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, plus the ability to access Secure or Non-secure physical memory is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 when EL3 is implemented.
- 0b0001 or 0b0010 when EL3 is not implemented.

This field is only valid when the value of ID_PFR1.Security is 0, otherwise it holds the value 0b0000.

### GenTimer, bits [19:16]

Generic Timer support. Defined values are:

<table>
<thead>
<tr>
<th>GenTimer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Generic Timer is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Generic Timer is implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Generic Timer is implemented, and also includes support for CNTHCTL.EVNTIS and CNTKCTL.EVNTIS fields, and CNTPCTSS and CNTVCVSS counter views.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.6-ECV implements the functionality identified by the value 0b0010.

From Armv8.0 to Armv8.4, the only permitted value is 0b0001.

From Armv8.6, the only permitted value is 0b0010.

### Virtualization, bits [15:12]

Virtualization support. Defined values are:
Virtualization

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL2, Hyp mode, and the HVC instruction not implemented.</td>
</tr>
<tr>
<td>EL2, Hyp mode, the HVC instruction, and all the features described by</td>
</tr>
<tr>
<td>Virt_frac == 0b0001 implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

Note

The ID_ISARs do not identify whether the HVC instruction is implemented.

MProgMod, bits [11:8]

M profile programmers' model support. Defined values are:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not supported.</td>
</tr>
<tr>
<td>Support for two-stack programmers' model.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Security, bits [7:4]

Security support. Defined values are:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>EL3, Monitor mode, and the SMC instruction not implemented.</td>
</tr>
<tr>
<td>EL3, Monitor mode, the SMC instruction, and all the features described</td>
</tr>
<tr>
<td>by Sec_frac == 0b0001 implemented.</td>
</tr>
<tr>
<td>As for 0b0001, and adds the ability to set the NSACR.RFR bit.</td>
</tr>
<tr>
<td>Not permitted in Armv8 as the NSACR.RFR bit is RES0.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- 0b0000 when EL3 is not implemented.
- 0b0001 when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

ProgMod, bits [3:0]

Support for the standard programmers' model for ARMv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes. Defined values are:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not supported.</td>
</tr>
<tr>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A the only permitted value is \(0b0001\).

### Accessing the ID_PFR1

Accesses to this register use the following encodings:

\[
\text{MRC}\{c\}\{q\} \langle\text{coproc}\rangle, \\#\langle\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle, \\#\langle\text{opc2}\rangle
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
      AArch32.TakeHypTrapException(0x03);
   else
      return ID_PFR1;
   endif
elsif PSTATE.EL == EL2 then
   return ID_PFR1;
elsif PSTATE.EL == EL3 then
   return ID_PFR1;
endif
ID_PFR2, Processor Feature Register 2

The ID_PFR2 characteristics are:

**Purpose**

Gives information about the AArch32 programmers' model.

Must be interpreted with ID_PFR0 and ID_PFR1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register ID_PFR2 bits [31:0] are architecturally mapped to AArch64 System register ID_PFR2_EL1[31:0].

**Attributes**

ID_PFR2 is a 32-bit register.

**Field descriptions**

The ID_PFR2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RAS_frac | SSBS | CSV3 |

**Bits [31:12]**

Reserved, RES0.

**RAS_frac, bits [11:8]**

From Armv8.4:

RAS Extension fractional field.

<table>
<thead>
<tr>
<th>RAS_frac</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>If ID_PFR0.RAS == 0b0001, RAS Extension implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>If ID_PFR0.RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC&lt;m&gt; System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR&lt;n&gt;STATUS and support for the optional RAS Timestamp Extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is valid only if ID_PFR0.RAS == 0b0001.

**Otherwise:**

Reserved, RES0.

**SSBS, bits [7:4]**
From Armv8.5:

Speculative Store Bypassing controls in AArch64 state. Defined values are:

<table>
<thead>
<tr>
<th>SSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>AArch32 provides no mechanism to control the use of Speculative Store Bypassing.</td>
</tr>
<tr>
<td>0b0001</td>
<td>AArch32 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe.</td>
</tr>
</tbody>
</table>

From Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

Otherwise:

Reserved, RES0.

CSV3, bits [3:0]

From Armv8.5:

Speculative use of faulting data. Defined values are:

<table>
<thead>
<tr>
<th>CSV3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>This Device does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by instructions newer than the load in the speculative sequence</td>
</tr>
<tr>
<td>0b0001</td>
<td>Data loaded under speculation with a permission or domain fault cannot be used to form an address or generate condition codes or SVE predicate values to be used by instructions newer than the load in the speculative sequence</td>
</tr>
</tbody>
</table>

From Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

Otherwise:

Reserved, RES0.

Accessing the ID_PFR2

Accesses to this register use the following encodings:

\[
\text{MRC} \{<c>\} \{<q>\} <\text{coproc}>, \{\#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{\#\}<\text{opc2}> \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_PFR2;
    endif
elsif PSTATE.EL == EL2 then
    return ID_PFR2;
elsif PSTATE.EL == EL3 then
    return ID_PFR2;

IFAR, Instruction Fault Address Register

The IFAR characteristics are:

**Purpose**

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception.

**Configuration**

AArch32 System register IFAR bits [31:0] are architecturally mapped to AArch64 System register [FAR_EL1][63:32].

AArch32 System register IFAR bits [31:0] (S) are architecturally mapped to AArch32 System register [HIFAR][31:0] when HaveEL(EL2), HaveEL(EL3) and HighestELUsingAArch32().

AArch32 System register IFAR bits [31:0] (S) are architecturally mapped to AArch64 System register [FAR_EL2][63:32] when HaveEL(EL2).

RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

IFAR is a 32-bit register.

**Field descriptions**

The IFAR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| **VA of faulting address of synchronous Prefetch Abort exception** |

**Bits [31:0]**

VA of faulting address of synchronous Prefetch Abort exception.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the IFAR**

Accesses to this register use the following encodings:

\[
\text{MRC}<c>\{<q>\} <coproc>, \{#<opc1>\}, <Rt>, <CRn>, <CRm>\{, \{#<opc2>\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return IFAR_S;
    else
      return IFAR_NS;
    end;
  else
    return IFAR;
  end;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return IFAR_NS;
  else
    return IFAR;
  end;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return IFAR_S;
  else
    return IFAR_NS;
  end;
else
  return IFAR NS;
end;

MCR{<c>}{<q>}{<coproc>}{<opc1>}{<Rt>}{<CRn>}{<CRm>}{<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      IFAR_S = R[t];
    else
      IFAR_NS = R[t];
    end;
  else
    IFAR = R[t];
  end;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    IFAR_NS = R[t];
  else
    IFAR = R[t];
  end;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    IFAR_S = R[t];
  else
    IFAR_NS = R[t];
  end;
IFSR, Instruction Fault Status Register

The IFSR characteristics are:

**Purpose**

Holds status information about the last instruction fault.

**Configuration**

AArch32 System register IFSR bits [31:0] are architecturally mapped to AArch64 System register IFSR32_EL2[31:0].

The current translation table format determines which format of the register is used.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

IFSR is a 32-bit register.

**Field descriptions**

The IFSR bit assignments are:

When TTBCR.EAE == 0:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     | RES0 | FnV | RES0 | ExT | RES0 | FS[4] | LPAE | RES0 | FS[3:0] |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [31:17]**

Reserved, RES0.

**FnV, bit [16]**

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>IFAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

This field resets to an architecturally UNKNOWN value.

**Bits [15:13]**

Reserved, RES0.

**ExT, bit [12]**

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.
This field resets to an architecturally **UNKNOWN** value.

**Bit [11]**

Reserved, **RES0**.

**FS[4], bit [10]**

This field is bit[4] of FS[4:0].


<table>
<thead>
<tr>
<th>FS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00001</td>
<td>PC alignment fault</td>
</tr>
<tr>
<td>0b00010</td>
<td>Debug exception</td>
</tr>
<tr>
<td>0b00011</td>
<td>Access flag fault, level 1</td>
</tr>
<tr>
<td>0b00101</td>
<td>Translation fault, level 1</td>
</tr>
<tr>
<td>0b00110</td>
<td>Access flag fault, level 2</td>
</tr>
<tr>
<td>0b00111</td>
<td>Translation fault, level 2</td>
</tr>
<tr>
<td>0b01000</td>
<td>Synchronous External abort, not on translation table walk</td>
</tr>
<tr>
<td>0b01001</td>
<td>Domain fault, level 1</td>
</tr>
<tr>
<td>0b01011</td>
<td>Domain fault, level 2</td>
</tr>
<tr>
<td>0b01100</td>
<td>Synchronous External abort, on translation table walk, level 1</td>
</tr>
<tr>
<td>0b01101</td>
<td>Permission fault, level 1</td>
</tr>
<tr>
<td>0b01110</td>
<td>Synchronous External abort, on translation table walk, level 2</td>
</tr>
<tr>
<td>0b01111</td>
<td>Permission fault, level 2</td>
</tr>
<tr>
<td>0b10000</td>
<td>TLB conflict abort</td>
</tr>
<tr>
<td>0b10100</td>
<td>IMPLEMENTATION DEFINED fault (Lockdown fault)</td>
</tr>
<tr>
<td>0b11001</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk</td>
</tr>
<tr>
<td>0b11100</td>
<td>Synchronous parity or ECC error on translation table walk, level 1</td>
</tr>
<tr>
<td>0b11110</td>
<td>Synchronous parity or ECC error on translation table walk, level 2</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b11001, 0b11100, and 0b11110, are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The FS field is split as follows:

- FS[4] is IFSR[10].
- FS[3:0] is IFSR[3:0].

This field resets to an architecturally **UNKNOWN** value.

**LPAE, bit [9]**

On taking a Data Abort exception, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table formats.</td>
</tr>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table formats.</td>
</tr>
</tbody>
</table>

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

This field resets to an architecturally **UNKNOWN** value.

**Bits [8:4]**

Reserved, **RES0**.
FS[3:0], bits [3:0]

This field is bits[3:0] of FS[4:0].

When TTBCR.EAE == 1:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>FnV</td>
<td>RES0</td>
<td>ExT</td>
<td>RES0</td>
<td>LPAE</td>
<td>RES0</td>
<td>STATUS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

<table>
<thead>
<tr>
<th>FnV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IFAR is valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>IFAR is not valid, and holds an UNKNOWN value.</td>
</tr>
</tbody>
</table>

This field is only valid for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.
This field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

This field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table formats.</td>
</tr>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table formats.</td>
</tr>
</tbody>
</table>

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

This field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.
## STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Address size fault in TTBR0 or TTBR1.</td>
</tr>
<tr>
<td>0b000001</td>
<td>Address size fault, level 1.</td>
</tr>
<tr>
<td>0b000010</td>
<td>Address size fault, level 2.</td>
</tr>
<tr>
<td>0b000011</td>
<td>Address size fault, level 3.</td>
</tr>
<tr>
<td>0b000100</td>
<td>Translation fault, level 1.</td>
</tr>
<tr>
<td>0b000101</td>
<td>Translation fault, level 2.</td>
</tr>
<tr>
<td>0b000110</td>
<td>Translation fault, level 3.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Access flag fault, level 1.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Access flag fault, level 2.</td>
</tr>
<tr>
<td>0b001100</td>
<td>Access flag fault, level 3.</td>
</tr>
<tr>
<td>0b001101</td>
<td>Permission fault, level 1.</td>
</tr>
<tr>
<td>0b001110</td>
<td>Permission fault, level 2.</td>
</tr>
<tr>
<td>0b001111</td>
<td>Permission fault, level 3.</td>
</tr>
<tr>
<td>0b010000</td>
<td>Synchronous External abort, not on translation table walk.</td>
</tr>
<tr>
<td>0b010101</td>
<td>Synchronous External abort, on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b010110</td>
<td>Synchronous External abort, on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b010111</td>
<td>Synchronous External abort, on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b011000</td>
<td>Synchronous parity or ECC error on memory access, not on translation table walk.</td>
</tr>
<tr>
<td>0b011101</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 1.</td>
</tr>
<tr>
<td>0b011110</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 2.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Synchronous parity or ECC error on memory access on translation table walk, level 3.</td>
</tr>
<tr>
<td>0b100001</td>
<td>PC alignment fault.</td>
</tr>
<tr>
<td>0b100100</td>
<td>Debug exception.</td>
</tr>
<tr>
<td>0b110000</td>
<td>TLB conflict abort.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When the RAS Extension is implemented, 0b011000, 0b011101, 0b011110, and 0b011111, are reserved.

For more information about the lookup level associated with a fault, see ‘The level associated with MMU faults on a Long-descriptor translation table lookup’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

## Accessing the IFSR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} <\text{coproc}>\{,\#<\text{opc1}>\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\{,\#<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return IFSR_S;
        else
            return IFSR_NS;
        end
    else
        return IFSR;
else
    return IFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFSR_NS;
    else
        return IFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return IFSR_S;
    else
        return IFSR_NS;
else
    if SCR.NS == '0' then
        IFSR_S = R[t];
    else
        IFSR_NS = R[t];
else
    IFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFSR_NS = R[t];
else
    IFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFSR_S = R[t];
else
    IFSR_NS = R[t];
else
    IFSR = R[t];

MCR{<c>{q}<coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm}{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return IFSR_S;
        else
            return IFSR_NS;
        end
    else
        return IFSR;
else
    return IFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFSR_NS;
    else
        return IFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return IFSR_S;
    else
        return IFSR_NS;
else
    if SCR.NS == '0' then
        IFSR_S = R[t];
    else
        IFSR_NS = R[t];
else
    IFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFSR_NS = R[t];
else
    IFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFSR_S = R[t];
else
    IFSR_NS = R[t];
else
    IFSR = R[t];
ISR, Interrupt Status Register

The ISR characteristics are:

**Purpose**

Shows the pending status of the IRQ, FIQ, or SError.

When executing at EL2, EL3, or Secure EL1, when SCR_EL3.EEL2 == 0b0, this shows the pending status of the physical interrupts.

When executing at Non-secure EL1, or at Secure EL1, when SCR_EL3.EEL2 == 0b01:

- If the HCR.{IMO,FMO,AMO} bit has a value of 1, the corresponding ISR.{I,F,A} bit shows the pending status of the virtual IRQ, FIQ, or SError.
- If the HCR.{IMO,FMO,AMO} bit has a value of 0, the corresponding ISR.{I,F,A} bit shows the pending status of the physical IRQ, FIQ, or SError.

**Configuration**

AArch32 System register ISR bits [31:0] are architecturally mapped to AArch64 System register **ISR_EL1[31:0]**.

**Attributes**

ISR is a 32-bit register.

**Field descriptions**

The ISR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Res0 | A | I | F | Res0 |

**Bits [31:9]**

Reserved, Res0.

**A, bit [8]**

SError interrupt pending bit:

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No pending SError interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>An SError interrupt is pending.</td>
</tr>
</tbody>
</table>

If the SError interrupt is edge-triggered, this field is cleared to zero when the physical SError interrupt is taken.

**I, bit [7]**

IRQ pending bit. Indicates whether an IRQ interrupt is pending:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No pending IRQ.</td>
</tr>
<tr>
<td>0b1</td>
<td>An IRQ interrupt is pending.</td>
</tr>
</tbody>
</table>
F, bit [6]

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No pending FIQ.</td>
</tr>
<tr>
<td>0b1</td>
<td>An FIQ interrupt is pending.</td>
</tr>
</tbody>
</table>

Bits [5:0]

Reserved, RES0.

Accessing the ISR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#<\text{opc2}>)
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return ISR;
elsif PSTATE.EL == EL2 then
  return ISR;
elsif PSTATE.EL == EL3 then
  return ISR;
ITLBIALL, Instruction TLB Invalidate All

The ITLBIALL characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from instruction TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

**Configuration**

**Attributes**

ITLBIALL is a 32-bit System instruction.

**Field descriptions**

ITLBIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the ITLBIALL instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} \ <\text{coproc}>, \{#\}<\text{opc1}>, \ <\text{Rt}>, \ <\text{CRn}>, \ <\text{CRm}>, \ {, \{#\}<\text{opc2}>}\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ITLBIALL();
    end if;
elsif PSTATE.EL == EL2 then
    ITLBIALL();
elsif PSTATE.EL == EL3 then
    ITLBIALL();
end if;
The ITLBIASID characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

**Configuration**

**Attributes**

ITLBIASID is a 32-bit System instruction.

**Field descriptions**

The ITLBIASID input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7-0</td>
<td>ASID</td>
<td>ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.</td>
</tr>
</tbody>
</table>

**Executing the ITLBIASID instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ITLBIASID(R[t]);
    end
elsif PSTATE.EL == EL2 then
    ITLBIASID(R[t]);
elsif PSTATE.EL == EL3 then
    ITLBIASID(R[t]);
else
    ITLBIASID(R[t]);

The ITLBIMVA characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

**Configuration**

**Attributes**

ITLBIMVA is a 32-bit System instruction.

**Field descriptions**

The ITLBIMVA input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VA |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| ASID |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:8]**

Reserved, RES0.

**ASID, bits [7:0]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this operation, regardless of the value of the ASID field.
Executing the ITLBIMVA instruction

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>}\ <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\{,#\}<\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        ITLBIAID(R[t]);
    end
elsif PSTATE.EL == EL2 then
    ITLBIAID(R[t]);
elsif PSTATE.EL == EL3 then
    ITLBIAID(R[t]);
**JIDR, Jazelle ID Register**

The JIDR characteristics are:

**Purpose**

A Jazelle register, which identified the Jazelle architecture version.

**Configuration**

**Attributes**

JIDR is a 32-bit register.

**Field descriptions**

The JIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RAZ</td>
</tr>
<tr>
<td>30-0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RAZ.

**Accessing the JIDR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\} {, \{#\}<\text{opc2}\} \\
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b111</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if boolean IMPLEMENTATION_DEFINED "JIDR UNDEFINED at EL0" then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '1' && HCR_EL2.TID0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
    AArch32.TakeHypTrapException(0x05);
  else
    return JIDR;
  endif
else
  return JIDR;
endif

if PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
    AArch32.TakeHypTrapException(0x05);
  else
    return JIDR;
  endif
else
  return JIDR;
endif

if PSTATE.EL == EL2 then
  return JIDR;
else
  return JIDR;
endif

if PSTATE.EL == EL3 then
  return JIDR;
JMCRI, Jazelle Main Configuration Register

The JMCRI characteristics are:

**Purpose**

A Jazelle register, which provides control of the Jazelle extension.

**Configuration**

**Attributes**

JMCRI is a 32-bit register.

**Field descriptions**

The JMCRI bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RAZ/WI.

**Accessing the JMCRI**

For accesses from EL0 it is IMPLEMENTATION DEFINED whether the register is RW or UNDEFINED.

Accesses to this register use the following encodings:

```plaintext
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b111</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION DEFINED "JMCRI UNDEFINED at EL0" then
        UNDEFINED;
    else
        return JMCRI;
elsif PSTATE.EL == EL1 then
    return JMCRI;
elsif PSTATE.EL == EL2 then
    return JMCRI;
elsif PSTATE.EL == EL3 then
    return JMCRI;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b111</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        //no operation
elsif PSTATE.EL == EL1 then
    //no operation
elsif PSTATE.EL == EL2 then
    //no operation
elsif PSTATE.EL == EL3 then
    //no operation
JOSCR, Jazelle OS Control Register

The JOSCR characteristics are:

**Purpose**

A Jazelle register, which provides operating system control of the Jazelle Extension.

**Configuration**

**Attributes**

JOSCR is a 32-bit register.

**Field descriptions**

The JOSCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RAZ/WI</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RAZ/WI.

**Accessing the JOSCR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b111</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
    UNDEFINED;
  else
    return JOSCR;
elsif PSTATE.EL == EL1 then
  return JOSCR;
elsif PSTATE.EL == EL2 then
  return JOSCR;
elsif PSTATE.EL == EL3 then
  return JOSCR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b111</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        //no operation
elseif PSTATE.EL == EL1 then
    //no operation
elseif PSTATE.EL == EL2 then
    //no operation
elseif PSTATE.EL == EL3 then
    //no operation
MAIR0, Memory Attribute Indirection Register 0

The MAIR0 characteristics are:

**Purpose**

Along with MAIR1, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations.

AttrIndx[2] indicates the MAIR register to be used:

- When AttrIndx[2] is 0, MAIR0 is used.
- When AttrIndx[2] is 1, MAIR1 is used.

**Configuration**

AArch32 System register MAIR0 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[31:0] when TTBCR.EAE == 1.

MAIR0 and PRRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in PRRR.
- When it is set to 1, the register is as described in MAIR0.

When EL3 is using AArch32, write access to MAIR0(S) is disabled when the CP15SDISABLE signal is asserted HIGH. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MAIR0 is a 32-bit register.

**Field descriptions**

The MAIR0 bit assignments are:

**When TTBCR.EAE == 1:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attr3</td>
<td>Attr2</td>
<td>Attr1</td>
<td>Attr0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Attr<n>, bits [8n+7:8n], for n = 0 to 3**

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

<table>
<thead>
<tr>
<th>Attr&lt;n&gt;[7:4]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device memory. See encoding of Attr&lt;n&gt;[3:0] for the type of Device memory.</td>
</tr>
<tr>
<td>0b00RW, RW</td>
<td>Normal memory, Outer Write-Through Transient.</td>
</tr>
<tr>
<td>not 0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Outer Write-Back Transient.</td>
</tr>
<tr>
<td>0b01RW, RW</td>
<td>Normal memory, Outer Write-Back Non-transient.</td>
</tr>
<tr>
<td>not 0b01</td>
<td>Normal memory, Outer Write-Through Non-transient.</td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Outer Write-Back Non-transient.</td>
</tr>
<tr>
<td>0b11RW</td>
<td>Normal memory, Outer Write-Back Non-transient.</td>
</tr>
</tbody>
</table>
R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

<table>
<thead>
<tr>
<th>Attr&lt;n&gt;[3:0]</th>
<th>Meaning when Attr&lt;n&gt;[7:4] is 0b0000</th>
<th>Meaning when Attr&lt;n&gt;[7:4] is not 0b0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Through Transient</td>
</tr>
<tr>
<td>0b00RW, RW</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Non-cacheable</td>
</tr>
<tr>
<td>not 0b00</td>
<td>Device-nGnRE memory</td>
<td>Normal memory, Inner Write-Back Transient</td>
</tr>
<tr>
<td>0b01RW, RW</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Through Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>not 0b00</td>
<td>Device-nGRE memory</td>
<td>Normal memory, Inner Write-Back Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>0b10RW, RW</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Through Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>not 0b00</td>
<td>Device-GRE memory</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
<tr>
<td>0b11RW, RW</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
<tr>
<td>not 0b00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

<table>
<thead>
<tr>
<th>R or W</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Allocate</td>
</tr>
<tr>
<td>0b1</td>
<td>Allocate</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the MAIR0**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return MAIR0_S;
        else
            return MAIR0_NS;
        end if
    end if
else
    return MAIR0;
else
    return MAIR0;
else if PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return MAIR0_NS;
    else
        return MAIR0;
elelse if PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return MAIR0_S;
    else
        return MAIR0_NS;
end if
end if

MCR{<c}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            MAIR0_S = R[t];
        else
            MAIR0_NS = R[t];
        end if
    else
        MAIR0 = R[t];
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        MAIR0_NS = R[t];
    else
        MAIR0 = R[t];
    end if
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            MAIR0_S = R[t];
        else
            MAIR0_NS = R[t];
        end if
    end if
MAIR1, Memory Attribute Indirection Register 1

The MAIR1 characteristics are:

**Purpose**

Along with MAIR0, provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations.

AttrIndx[2] indicates the MAIR register to be used:

- When AttrIndx[2] is 0, MAIR0 is used.
- When AttrIndx[2] is 1, MAIR1 is used.

**Configuration**

AArch32 System register MAIR1 bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[63:32] when TTBCR.EAE == 1.

MAIR1 and NMRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in NMRR.
- When it is set to 1, the register is as described in MAIR1.

When EL3 is using AArch32, write access to MAIR1(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

MAIR1 is a 32-bit register.

**Field descriptions**

The MAIR1 bit assignments are:

When TTBCR.EAE == 1:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Attr7 | Attr6 | Attr5 | Attr4 |

**Attr<n>, bits [8(n-4)+7:8(n-4)], for n = 4 to 7**

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

<table>
<thead>
<tr>
<th>Attr&lt;n&gt;[7:4]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device memory. See encoding of Attr&lt;n&gt;[3:0] for the type of Device memory.</td>
</tr>
<tr>
<td>0b00RW, RW</td>
<td>Normal memory, Outer Write-Through Transient.</td>
</tr>
<tr>
<td>not 0b00</td>
<td></td>
</tr>
<tr>
<td>0b0100</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01RW, RW</td>
<td>Normal memory, Outer Write-Back Transient.</td>
</tr>
<tr>
<td>not 0b01</td>
<td></td>
</tr>
<tr>
<td>0b10RW</td>
<td>Normal memory, Outer Write-Through Non-transient.</td>
</tr>
<tr>
<td>0b11RW</td>
<td>Normal memory, Outer Write-Back Non-transient.</td>
</tr>
</tbody>
</table>
R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

<table>
<thead>
<tr>
<th>Attr&lt;n&gt;[3:0]</th>
<th>Meaning when Attr&lt;n&gt;[7:4] is 0b0000</th>
<th>Meaning when Attr&lt;n&gt;[7:4] is not 0b0000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-nGnRnE memory</td>
<td>UNPREDICTABLE</td>
</tr>
<tr>
<td>0b000RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Through Transient</td>
</tr>
<tr>
<td>0b0100</td>
<td>Device-nGnRE memory</td>
<td>Normal memory, Inner Non-cacheable</td>
</tr>
<tr>
<td>0b01RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Transient</td>
</tr>
<tr>
<td>0b1000</td>
<td>Device-nGRE memory</td>
<td>Normal memory, Inner Write-Through Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>0b10RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>0b1100</td>
<td>Device-GRE memory</td>
<td>Normal memory, Inner Write-Through Non-transient (RW=0b00)</td>
</tr>
<tr>
<td>0b11RW, RW not 0b00</td>
<td>UNPREDICTABLE</td>
<td>Normal memory, Inner Write-Back Non-transient</td>
</tr>
</tbody>
</table>

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

<table>
<thead>
<tr>
<th>R or W</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Allocate</td>
</tr>
<tr>
<td>0b1</td>
<td>Allocate</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the MAIR1**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}\ <\text{coproc}, \{#\}<\text{opc1}>, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}\{, \{#\}<\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return MAIR1_S;
        else
            return MAIR1_NS;
        end;
    else
        return MAIR1;
    end;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return MAIR1_NS;
    else
        return MAIR1;
    end;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return MAIR1_S;
    else
        return MAIR1_NS;
end;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            MAIR1_S = R[t];
        else
            MAIR1_NS = R[t];
        end
    else
        MAIR1 = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        MAIR1_NS = R[t];
    else
        MAIR1 = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            MAIR1_S = R[t];
        else
            MAIR1_NS = R[t];
        end
    end

The MIDR characteristics are:

**Purpose**

Provides identification information for the PE, including an implementer code for the device and a device ID number.

**Configuration**

AArch32 System register MIDR bits [31:0] are architecturally mapped to AArch64 System register `MIDR_EL1[31:0]`.

AArch32 System register MIDR bits [31:0] are architecturally mapped to External register `MIDR_EL1[31:0]`.

Some fields of the MIDR are IMPLEMENTATION DEFINED. For details of the values of these fields for a particular Armv8 implementation, and any implementation-specific significance of these values, see the product documentation.

**Attributes**

MIDR is a 32-bit register.

**Field descriptions**

The MIDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Implementer | Variant | Architecture | PartNum | Revision |

**Implementer, bits [31:24]**

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

<table>
<thead>
<tr>
<th>Hex representation</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved for software use</td>
</tr>
<tr>
<td>0xC0</td>
<td>Ampere Computing</td>
</tr>
<tr>
<td>0x41</td>
<td>Arm Limited</td>
</tr>
<tr>
<td>0x42</td>
<td>Broadcom Corporation</td>
</tr>
<tr>
<td>0x43</td>
<td>Cavium Inc.</td>
</tr>
<tr>
<td>0x44</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>0x46</td>
<td>Fujitsu Ltd.</td>
</tr>
<tr>
<td>0x49</td>
<td>Infineon Technologies AG</td>
</tr>
<tr>
<td>0x40</td>
<td>Motorola or Freescale Semiconductor Inc.</td>
</tr>
<tr>
<td>0x4E</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>0x50</td>
<td>Applied Micro Circuits Corporation</td>
</tr>
<tr>
<td>0x51</td>
<td>Qualcomm Inc.</td>
</tr>
<tr>
<td>0x56</td>
<td>Marvell International Ltd.</td>
</tr>
<tr>
<td>0x69</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

**Variant, bits [23:20]**

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.
The permitted values of this field are:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Armv4.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Armv4T.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Armv5 (obsolete).</td>
</tr>
<tr>
<td>0b0100</td>
<td>Armv5I.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Armv5TE.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Armv5TEJ.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv6.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Architectural features are individually identified in the ID.* registers, see 'ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K12.5.3.</td>
</tr>
</tbody>
</table>

All other values are reserved.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

Accessing the MIDR

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) then
    return VPIDR_EL2<31:0>;
  elsif EL2Enabled() && ELUsingAAArch32(EL2) then
    return VPIDR;
  else
    return MIDR;
elsif PSTATE.EL == EL2 then
  return MIDR;
else
  return MIDR;
```
**MPI DR, Multiprocessor Affinity Register**

The MPIDR characteristics are:

**Purpose**

In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

**Configuration**

AArch32 System register MPIDR bits [31:0] are architecturally mapped to AArch64 System register `MPIDR_EL1[31:0]`.

In a uniprocessor system Arm recommends that each Aff<n> field of this register returns a value of 0.

**Attributes**

MPI DR is a 32-bit register.

**Field descriptions**

The MPIDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| M  | U  | RES0| MT | Aff2|    | Aff1|    | Aff0|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**M, bit [31]**

Indicates whether this implementation includes the functionality introduced by the ARMv7 Multiprocessing Extensions. The possible values of this bit are:

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This implementation does not include the ARMv7 Multiprocessing Extensions functionality.</td>
</tr>
<tr>
<td>0b1</td>
<td>This implementation includes the ARMv7 Multiprocessing Extensions functionality.</td>
</tr>
</tbody>
</table>

From Armv8, this bit is RAO.

**U, bit [30]**

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system. The possible values of this bit are:

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Processor is part of a multiprocessor system.</td>
</tr>
<tr>
<td>0b1</td>
<td>Processor is part of a uniprocessor system.</td>
</tr>
</tbody>
</table>

**Bits [29:25]**

Reserved, RES0.

**MT, bit [24]**

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels. The possible values of this bit are:
<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Performance of PEs at the lowest affinity level, or PEs with MPIDR.MT set to 1, different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent.</td>
</tr>
<tr>
<td>0b1</td>
<td>Performance of PEs at the lowest affinity level, or PEs with MPIDR.MT set to 1, different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent.</td>
</tr>
</tbody>
</table>

**Aff2, bits [23:16]**

Affinity level 2. See the description of Aff0 for more information.

**Aff1, bits [15:8]**

Affinity level 1. See the description of Aff0 for more information.

**Aff0, bits [7:0]**

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or MPIDR_EL1.{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

**Accessing the MPIDR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) then
    return VMPIDR_EL2<31:0>;
  elsif EL2Enabled() && ELUsingAArch32(EL2) then
    return VMPIDR;
  else
    return MPIDR;
elsif PSTATE.EL == EL2 then
  return MPIDR;
elsif PSTATE.EL == EL3 then
  return MPIDR;
MVBAR, Monitor Vector Base Address Register

The MVBAR characteristics are:

**Purpose**

When EL3 is implemented and can use AArch32, holds the vector base address for any exception that is taken to Monitor mode.

Secure software must program the MVBAR with the required initial value as part of the PE boot sequence.

**Configuration**

It is **IMPLEMENTATION DEFINED** whether MVBAR[0] has a fixed value and ignored writes, or takes the last value written to it.

On a reset into EL3 using AArch32, the reset value of MVBAR is an **IMPLEMENTATION DEFINED** choice between:

- MVBAR[31:5] = a **IMPLEMENTATION DEFINED** value, which might be **UNKNOWN**.
- MVBAR[4:1] = **RES**0.
- MVBAR[0] = 0.

And:

- MVBAR[31:1] = a **IMPLEMENTATION DEFINED** value that is bits[31:1] of the AArch32 reset address.
- MVBAR[0] = 1.

**Attributes**

MVBAR is a 32-bit register.

**Field descriptions**

The MVBAR bit assignments are:

**When programmed with a vector base address:**

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  |  9  |  8  |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Vector Base Address**

**Reserved**

**Bits [31:5]**

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

**Reserved, bits [4:0]**

Reserved, see Configurations.

**Accessing the MVBAR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{\langle c\rangle\}\{\langle q\rangle\} \langle\text{coproc}\rangle, \{\#\}\langle\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle\{, \{\#\}\langle\text{opc2}\rangle\}
\]

| coproc | opc1 | CRn | CRm | opc2 |
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
        return MVBAR;
    endif
fi

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' && CP15SDISABLE == HIGH then
            UNDEFINED;
        elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
            UNDEFINED;
        else
            MVBAR = R[t];
        endif
    endif
fi

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The MVFR0 characteristics are:

**Purpose**

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with MVFR1 and MVFR2.

For general information about the interpretation of the ID registers see ‘Principles of the ID scheme for fields in ID registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register MVFR0 bits [31:0] are architecturally mapped to AArch64 System register MVFR0_EL1[31:0].

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

**Attributes**

MVFR0 is a 32-bit register.

**Field descriptions**

The MVFR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| FPRound | FPShVec | FPSqr | FPDivide | FPTrap | FPDP | FPSP | SIMDReg |

**FPRound, bits [31:28]**

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes. Defined values are:

<table>
<thead>
<tr>
<th>FPRound</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the FPSCR setting.</td>
</tr>
<tr>
<td>0b0001</td>
<td>All rounding modes supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

**FPShVec, bits [27:24]**

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors. Defined values are:

<table>
<thead>
<tr>
<th>FPShVec</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Short vectors not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Short vector operation supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.
**FPSqrt, bits [23:20]**

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations. Defined values are:

<table>
<thead>
<tr>
<th>FPSqrt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

**FPDivide, bits [19:16]**

Indicates whether the floating-point implementation provides support for VFP divide operations. Defined values are:

<table>
<thead>
<tr>
<th>FPDivide</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

**FPTrap, bits [15:12]**

Floating Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping. Defined values are:

<table>
<thead>
<tr>
<th>FPTrap</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0001</td>
<td>Supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

A value of 0b0001 indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

**FPDP, bits [11:8]**

Double Precision. Indicates whether the floating-point implementation provides support for double-precision operations. Defined values are:

<table>
<thead>
<tr>
<th>FPDP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0001</td>
<td>Supported, VFPv2.</td>
</tr>
<tr>
<td>0010</td>
<td>Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is 0b0001.
- VDIV.F64 is only available if the Divide field is 0b0001.
Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

**FPSP, bits [7:4]**

Single Precision. Indicates whether the floating-point implementation provides support for single-precision operations. Defined values are:

<table>
<thead>
<tr>
<th>FPSP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported in hardware.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Supported, VFPv2.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

**SIMDReg, bits [3:0]**

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank. Defined values are:

<table>
<thead>
<tr>
<th>SIMDReg</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The implementation has no Advanced SIMD and floating-point support.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation includes floating-point support with 16 x 64-bit registers.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

**Accessing the MVFR0**

Accesses to this register use the following encodings:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

<table>
<thead>
<tr>
<th>reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0111</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00') then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPtr.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x08);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x08);
  else
    return MVFR0;
  elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPtr.TCP10 == '1') then
      AArch32.TakeHypTrapException(0x00);
    else
      return MVFR0;
  elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
      UNDEFINED;
    else
      return MVFR0;
  end if
else
return MVFR0;
end if
The MVFR1 characteristics are:

### Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation. Must be interpreted with MVFR0 and MVFR2.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

### Configuration

AArch32 System register MVFR1 bits [31:0] are architecturally mapped to AArch64 System register MVFR1_EL1[31:0].

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

### Attributes

MVFR1 is a 32-bit register.

### Field descriptions

The MVFR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>SIMDFMAC</td>
</tr>
<tr>
<td>30</td>
<td>FPHP</td>
</tr>
<tr>
<td>29</td>
<td>SIMDHP</td>
</tr>
<tr>
<td>28</td>
<td>SIMDSP</td>
</tr>
<tr>
<td>27</td>
<td>SIMDInt</td>
</tr>
<tr>
<td>26</td>
<td>SIMDLS</td>
</tr>
<tr>
<td>25</td>
<td>FPDNaN</td>
</tr>
<tr>
<td>24</td>
<td>FPftZ</td>
</tr>
</tbody>
</table>

#### SIMDFMAC, bits [31:28]

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDFMAC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.

#### FPHP, bits [27:24]

Floating Point Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

<table>
<thead>
<tr>
<th>FPHP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds instructions for conversion between double-precision and half-precision.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As for 0b0010, and adds support for half-precision floating-point arithmetic.</td>
</tr>
</tbody>
</table>
All other values are reserved.

In Armv8-A the permitted values are:

- \(0b0000\) in an implementation without floating-point support.
- \(0b0010\) in an implementation with floating-point support that does not include the ARMv8.2-FP16 extension.
- \(0b0011\) in an implementation with floating-point support that includes the ARMv8.2-FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

<table>
<thead>
<tr>
<th>Half Precision instructions supported</th>
<th>FPHP</th>
<th>SIMDHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>No support</td>
<td>0b0000</td>
<td>0b0000</td>
</tr>
<tr>
<td>Conversions only</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
<tr>
<td>Conversions and arithmetic</td>
<td>0b0011</td>
<td>0b0010</td>
</tr>
</tbody>
</table>

**SIMDHP, bits [23:20]**

Advanced SIMD Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

<table>
<thead>
<tr>
<th>SIMDHP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As for 0b0001, and adds support for half-precision floating-point arithmetic.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are:

- \(0b0000\) in an implementation without SIMD floating-point support.
- \(0b0010\) in an implementation with SIMD floating-point support that does not include the ARMv8.2-FP16 extension.
- \(0b0011\) in an implementation with SIMD floating-point support that includes the ARMv8.2-FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

<table>
<thead>
<tr>
<th>Half Precision instructions supported</th>
<th>FPHP</th>
<th>SIMDHP</th>
</tr>
</thead>
<tbody>
<tr>
<td>No support</td>
<td>0b0000</td>
<td>0b0000</td>
</tr>
<tr>
<td>Conversions only</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
<tr>
<td>Conversions and arithmetic</td>
<td>0b0011</td>
<td>0b0010</td>
</tr>
</tbody>
</table>

**SIMDSP, bits [19:16]**

Advanced SIMD Single Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDSP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented. This value is permitted only if the SIMDInt field is 0b0001.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

**SIMDInt, bits [15:12]**

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDInt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented.</td>
</tr>
</tbody>
</table>

Page 2818
All other values are reserved.
In Armv8-A the permitted values are 0b0000 and 0b0001.

**SIMDLS, bits [11:8]**

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions. Defined values are:

<table>
<thead>
<tr>
<th>SIMDLS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A the permitted values are 0b0000 and 0b0001.

**FPDNaN, bits [7:4]**

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode. Defined values are:

<table>
<thead>
<tr>
<th>FPDNaN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or hardware supports only the Default NaN mode.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Hardware supports propagation of NaN values.</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A the permitted values are 0b0000 and 0b0001.

**FPFtZ, bits [3:0]**

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation. Defined values are:

<table>
<thead>
<tr>
<th>FPFtZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or hardware supports only the Flush-to-Zero mode of operation.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Hardware supports full denormalized number arithmetic.</td>
</tr>
</tbody>
</table>

All other values are reserved.
In Armv8-A the permitted values are 0b0000 and 0b0001.

**Accessing the MVFR1**

Accesses to this register use the following encodings:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

<table>
<thead>
<tr>
<th>reg</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0b0110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
    CPACR.cp10 == '00') then
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
    AArch32.TakeHypTrapException(0x08);
  else
    return MVFR1;
  end;
elsif PSTATE.EL == EL2 then
  if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x07);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
    NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
    AArch32.TakeHypTrapException(0x07);
  elsif EL2Enabled() && HCR_EL2.TID3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x08);
  else
    return MVFR1;
  end;
elsif PSTATE.EL == EL3 then
  if CPACR.cp10 == '00' then
    UNDEFINED;
  else
    return MVFR1;
  end;
else
  return MVFR1;
end;
MVFR2, Media and VFP Feature Register 2

The MVFR2 characteristics are:

**Purpose**

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with MVFR0 and MVFR1.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G7.1.3.

**Configuration**

AArch32 System register MVFR2 bits [31:0] are architecturally mapped to AArch64 System register MVFR2_EL1[31:0].

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

**Attributes**

MVFR2 is a 32-bit register.

**Field descriptions**

The MVFR2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    | FPMisc |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    | SiMDMisc |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:8]**

Reserved, RES0.

**FPMisc, bits [7:4]**

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

<table>
<thead>
<tr>
<th>FPMisc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or no support for miscellaneous features.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for Floating-point selection.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As 0b0010, and Floating-point Round to Integer Floating-point.</td>
</tr>
<tr>
<td>0b0100</td>
<td>As 0b0011, and Floating-point MaxNum and MinNum.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0100.

**SiMDMisc, bits [3:0]**

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.
<table>
<thead>
<tr>
<th>SIMDMisc</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Not implemented, or no support for miscellaneous features.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Floating-point Conversion to Integer with Directed Rounding modes.</td>
</tr>
<tr>
<td>0b0010</td>
<td>As 0b0001, and Floating-point Round to Integer Floating-point.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As 0b0010, and Floating-point MaxNum and MinNum.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0011.

**Accessing the MVFR2**

Accesses to this register use the following encodings:

\[
\text{VMRS} \{<c>\} \{<q>\} <Rt>, <\text{spec_reg}>
\]

<table>
<thead>
<tr>
<th>reg</th>
<th>0b0101</th>
</tr>
</thead>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00') then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x08);
    else
        return MVFR2;
    end
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elseif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    else
        return MVFR2;
    end
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        return MVFR2;
    end
```

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
NMRR, Normal Memory Remap Register

The NMRR characteristics are:

**Purpose**

Provides additional mapping controls for memory regions that are mapped as Normal memory by their entry in the PRRR.

Used in conjunction with the PRRR.

**Configuration**

AArch32 System register NMRR bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[63:32] when TTBCR.EAE == 0.

MAIR1 and NMRR are the same register, with a different view depending on the value of TTBCR.EAE:
- When it is set to 0, the register is as described in NMRR.
- When it is set to 1, the register is as described in MAIR1.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

NMRR is a 32-bit register.

**Field descriptions**

The NMRR bit assignments are:

**When TTBCR.EAE == 0:**

<table>
<thead>
<tr>
<th>OR&lt;n&gt;, bits [2n+17:2n+16]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Region is Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Region is Write-Back, Write-Allocate.</td>
</tr>
<tr>
<td>0b10</td>
<td>Region is Write-Through, no Write-Allocate.</td>
</tr>
<tr>
<td>0b11</td>
<td>Region is Write-Back, no Write-Allocate.</td>
</tr>
</tbody>
</table>

The meaning of the field with n = 6 is IMPLEMENTATIONDEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATIONDEFINED.

This field resets to an architecturally UNKNOWN value.

**IR<n>, bits [2n+1:2n], for n = 0 to 7**

Inner Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated. The possible values of this field are:
The meaning of the field with \( n = 6 \) is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination \( \{ \text{TEX}[0] = 1, \ C = 1, \ B = 0 \} \) is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

## Accessing the NMRR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} \ <\text{coproc}> \ , \ {#}<\text{opc1}> \ , \ <\text{Rt}> \ , \ <\text{CRn}> \ , \ <\text{CRm}>\{, \ {#}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>

\[
\text{if} \ P\text{STATE.EL} == \text{EL0} \ then
\]

UNDEFINED;

elsif \ P\text{STATE.EL} == \text{EL1} \ then

if EL2Enabled() \& \& !ELUsingAArch32(EL2) \& \& HSTR_EL2.T10 == '1' then

AArch64.AArch32SystemAccessTrap(EL2, 0x03);

elsif EL2Enabled() \& \& ELUsingAArch32(EL2) \& \& HSTR.T10 == '1' then

AArch32.TakeHypTrapException(0x03);

elsif EL2Enabled() \& \& ELUsingAArch32(EL2) \& \& HCR_EL2.TRVM == '1' then

AArch64.AArch32SystemAccessTrap(EL2, 0x03);

elsif EL2Enabled() \& \& ELUsingAArch32(EL2) \& \& HCR.TRVM == '1' then

AArch32.TakeHypTrapException(0x03);

elsif HaveEL(EL3) \& \& ELUsingAArch32(EL3) then

if SCR.NS == '0' then

return NMRR_S;

else

return NMRR_NS;

else

return NMRR;

elsif \ P\text{STATE.EL} == \text{EL2} \ then

if HaveEL(EL3) \& \& ELUsingAArch32(EL3) then

return NMRR_NS;

else

return NMRR;

elsif \ P\text{STATE.EL} == \text{EL3} \ then

if SCR.NS == '0' then

return NMRR_S;

else

return NMRR_NS;

else

return NMRR_NS;

\[
\text{MCR}\{<c>\}\{<q>\} \ <\text{coproc}> \ , \ {#}<\text{opc1}> \ , \ <\text{Rt}> \ , \ <\text{CRn}> \ , \ <\text{CRm}>\{, \ {#}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            NMRR_S = R[t];
        else
            NMRR_NS = R[t];
        end
    else
        NMRR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        NMRR_NS = R[t];
    else
        NMRR_S = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            NMRR_S = R[t];
        else
            NMRR_NS = R[t];
        end
    end
NSACR, Non-Secure Access Control Register

The NSACR characteristics are:

### Purpose

When EL3 is implemented and can use AArch32, defines the Non-secure access permissions to Trace, Advanced SIMD and floating-point functionality. Also includes implementation defined bits that can define Non-secure access permissions for implementation defined functionality.

### Configuration

**Note**

In AArch64 state, the NSACR controls are replaced by controls in CPTR_EL3.

Some or all RW fields of this register have defined reset values. These apply whenever the register is accessible. This means they apply when the PE resets into EL3 using AArch32.

### Attributes

NSACR is a 32-bit register.

### Field descriptions

The NSACR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | NSTRCDIS | RES0 | IMPLEMENTATION DEFINED | NSASEDIS | RES0 | cp11 | cp10 | RES0 |

If EL3 is implemented and is using AArch64 then:

- Any read of the NSACR from Non-secure EL2 or Non-secure EL1 returns a value of 0x00000C00.
- Any read or write to NSACR from Secure EL1 is trapped as an exception to EL3.

If EL3 is not implemented, then any read of the NSACR from EL2 or EL1 returns a value of 0x00000C00.

**Bits [31:21]**

Reserved, RES0.

**NSTRCDIS, bit [20]**

Disables Non-secure System register accesses to all implemented trace registers.

<table>
<thead>
<tr>
<th>NSTRCDIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on:</td>
</tr>
<tr>
<td></td>
<td>• System register access to implemented trace registers.</td>
</tr>
<tr>
<td></td>
<td>• The behavior of CPACR_TRCDIS and HCPTR_TTA.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure System register accesses to all implemented trace registers are disabled, meaning:</td>
</tr>
<tr>
<td></td>
<td>• CPACR.TRCDIS behaves as RAO/WI in Non-secure state, regardless of its actual value.</td>
</tr>
<tr>
<td></td>
<td>• HCPTR.TTA behaves as RAO/WI, regardless of its actual value.</td>
</tr>
</tbody>
</table>
The implementation of this field must correspond to the implementation of the CPACR.TRCDIS field:

- If CPACR.TRCDIS is RAZ/WI, this field is RAZ/WI.
- If CPACR.TRCDIS is RW, this field is RW.

**Note**
- The ETMv4 architecture does not permit EL0 to access the trace registers. If the implementation includes an ETMv4 implementation, EL0 accesses to the trace registers are UNDEFINED.
- The architecture does not provide Non-secure access controls on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

In a system where the PE resets into EL3, this field resets to 0.

**Bit [19]**

Reserved, RES0.

**IMPLEMENTATION DEFINED, bits [18:16]**

IMPLEMENTATION DEFINED.

**NSASEDIS, bit [15]**

Disables Non-secure access to the Advanced SIMD functionality.

<table>
<thead>
<tr>
<th>NSASEDIS</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0      | This control has no effect on:  
|          | - Non-secure access to Advanced SIMD functionality.  
|          | - The behavior of CPACR.ASEDIS and HCPTR.TASE.  
| 0b1      | Non-secure access to the Advanced SIMD functionality is disabled, meaning:  
|          | - CPACR.ASEDIS behaves as RAO/WI in Non-secure state, regardless of its actual value.  
|          | - HCPTR.TASE behaves as RAO/WI, regardless of its actual value.  

The implementation of this field must correspond to the implementation of the CPACR.ASEDIS field:

- If CPACR.ASEDIS is RES0, this field is RES0. If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.
- If CPACR.ASEDIS is RAZ/WI, this field is RAZ/WI.
- If CPACR.ASEDIS is RW, this field is RW.

In a system where the PE resets into EL3, this field resets to 0.

**Bits [14:12]**

Reserved, RES0.

**cp11, bit [11]**

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the NSACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.
Enable Non-secure access to the Advanced SIMD and floating-point features. Possible values of the fields are:

<table>
<thead>
<tr>
<th>cp10</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0  | Advanced SIMD and floating-point features can be accessed only from Secure state. Any attempt to access this functionality from Non-secure state is UNDEFINED. When the PE is in Non-secure state:  
  - The CPACR (cp11, cp10) fields ignore writes and read as 0b00, access denied.  
  - The HCPTR (TCP11, TCP10) fields behave as RAO/WI, regardless of their actual values. |
| 0b1  | Advanced SIMD and floating-point features can be accessed from both Security states. |

If Non-secure access to the Advanced SIMD and floating-point functionality is enabled, the CPACR must be checked to determine the level of access that is permitted.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

**Bits [9:0]**

Reserved, RES0.

**Accessing the NSACR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if !EL2Enabled() & & !ELUsingAArch32(EL2) & & HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !EL2Enabled() & & ELUsingAArch32(EL2) & & HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAAArch32(EL2) & & SCR_EL3.<NS,EL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAAArch32(EL3) & & SCR_EL3.NS == '0' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  elsif !HaveEL(EL3) || (!ELUsingAAArch32(EL3) & & SCR_EL3.NS == '1') then
    return Zeros(20):'1100':Zeros(8);
  else
    return NSACR;
elsif PSTATE.EL == EL2 then
  if !HaveEL(EL3) || (!ELUsingAAArch32(EL3) & & SCR_EL3.NS == '1') then
    return Zeros(20):'1100':Zeros(8);
  else
    return NSACR;
elsif PSTATE.EL == EL3 then
  return NSACR;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
    UNDEFINED;
  else
    NSACR = R[t];
end if;
PAR, Physical Address Register

The PAR characteristics are:

**Purpose**

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

**Configuration**

AArch32 System register PAR bits [63:0] are architecturally mapped to AArch64 System register PAR_EL1[63:0].

PAR is accessed as a 32-bit value:

- When the PE is not in Hyp mode and is using the Short-descriptor translation table format.
- When the PE is in Hyp mode and executes an ATS12NSOPR, ATS12NSOPW, ATS12NSOUR, or ATS12NSOUW instruction and the value of HCR.VM is 0 and the value of TTBCR.EAE is 0.

In these cases, PAR[63:32] is RES0.

Otherwise, the PAR is accessed as a 64-bit value, if any of the following is true:

- When using the Long-descriptor translation table format.
- If the stage 1 address translation is disabled and TTBCR.EAE is set to 1.
- In an implementation that includes EL2, for the result of an ATS1Cxx instruction performed from Hyp mode.

For PL1&0 stage 1 translations, TTBCR.EAE selects the translation table format.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits[31:0] and do not modify bits[63:32].

The Configurations section specifies the cases where each PAR format is used.

**Field descriptions**

The PAR bit assignments are:

**When the instruction returned a 32-bit value to the PAR, PAR.F==0:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>PA</td>
<td>LPAE</td>
<td>NOS</td>
<td>NS</td>
<td>IMPLEMENTATION DEFINED</td>
<td>SH</td>
<td>Inner[2:0]</td>
<td>Outer[1:0]</td>
<td>SS</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR can return a value that indicates the resulting attributes, rather than the values that appear in the translation table descriptors. More precisely:

- Memory attribute fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the translation table descriptors. This applies to the NOS, SH, Inner, and Outer fields.
- See the NS bit description for constraints on the value it returns.
**Bits [63:32]**

Reserved, RES0.

**PA, bits [31:12]**

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[31:12].

This field resets to an architecturally **UNKNOWN** value.

**LPAE, bit [11]**

When updating the PAR with the result of the translation operation, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Short-descriptor translation table format used. This means the PAR returned a 32-bit value.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**NOS, bit [10]**

Not Outer Shareable. When the returned value of PAR.SH is 1, indicates the Shareability attribute for the physical memory region:

<table>
<thead>
<tr>
<th>NOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory region is Outer Shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory region is Inner Shareable.</td>
</tr>
</tbody>
</table>

When the returned value of PAR.SH is 0 the value returned to this field is **UNKNOWN**.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally **UNKNOWN** value.

**NS, bit [9]**

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is **UNKNOWN**.

This field resets to an architecturally **UNKNOWN** value.

**IMPLEMENTATION DEFINED, bit [8]**

**IMPLEMENTATION DEFINED.**

This field resets to an architecturally **UNKNOWN** value.

**SH, bit [7]**

Shareability. Indicates whether the physical memory region is Non-shareable:

<table>
<thead>
<tr>
<th>SH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory is Non-shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is shareable, and PAR.NOS indicates whether the region is Outer Shareable or Inner Shareable.</td>
</tr>
</tbody>
</table>
The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally UNKNOWN value.

**Inner[2:0], bits [6:4]**

Inner cacheability attribute for the region. Permitted values are:

<table>
<thead>
<tr>
<th>Inner[2:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Non-cacheable.</td>
</tr>
<tr>
<td>0b001</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b011</td>
<td>Device-nGnRE.</td>
</tr>
<tr>
<td>0b101</td>
<td>Write-Back, Write-Allocate.</td>
</tr>
<tr>
<td>0b110</td>
<td>Write-Through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Write-Back, no Write-Allocate.</td>
</tr>
</tbody>
</table>

The values 0b010 and 0b100 are reserved.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally UNKNOWN value.

**Outer[1:0], bits [3:2]**

Outer cacheability attribute for the region. Permitted values are:

<table>
<thead>
<tr>
<th>Outer[1:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Write-Back, Write-Allocate.</td>
</tr>
<tr>
<td>0b10</td>
<td>Write-Through, no Write-Allocate.</td>
</tr>
<tr>
<td>0b11</td>
<td>Write-Back, no Write-Allocate.</td>
</tr>
</tbody>
</table>

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally UNKNOWN value.

**SS, bit [1]**

Supersection. Used to indicate if the result is a Supersection:

<table>
<thead>
<tr>
<th>SS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Result is not a Supersection. PAR[31:12] contains OA[31:12].</td>
</tr>
<tr>
<td>0b1</td>
<td>Result is a Supersection, and:</td>
</tr>
<tr>
<td></td>
<td>• PAR[31:24] contains OA[31:24].</td>
</tr>
<tr>
<td></td>
<td>• PAR[15:12] contains 0b0000.</td>
</tr>
<tr>
<td></td>
<td>If an implementation supports less than 40 bits of physical address,</td>
</tr>
<tr>
<td></td>
<td>the bits in the PAR field that correspond to physical address bits</td>
</tr>
<tr>
<td></td>
<td>that are not implemented are UNKNOWN.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**F, bit [0]**

Indicates whether the instruction performed a successful address translation.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Address translation completed successfully.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
When the instruction returned a 32-bit value to the PAR, PAR.F==1:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED, bits [31:16]</td>
</tr>
<tr>
<td>61</td>
<td>IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>60</td>
<td>This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>59</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>58</td>
<td>LPAE, bit [11]</td>
</tr>
<tr>
<td>57</td>
<td>When updating the PAR with the result of the translation operation, this bit is set as follows:</td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>LPAE</td>
</tr>
<tr>
<td>54</td>
<td>0b0</td>
</tr>
<tr>
<td>53</td>
<td>0b1</td>
</tr>
<tr>
<td>52</td>
<td>This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>FS, bits [6:1]</td>
</tr>
<tr>
<td>49</td>
<td>Fault status bits. Bits [12,10,3:0] from the DFSR, indicating the source of the abort.</td>
</tr>
<tr>
<td>48</td>
<td>This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>47</td>
<td>F, bit [0]</td>
</tr>
<tr>
<td>46</td>
<td>Indicates whether the instruction performed a successful address translation.</td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>0b1</td>
</tr>
<tr>
<td>43</td>
<td>This field resets to an architecturally UNKNOWN value.</td>
</tr>
</tbody>
</table>

When the instruction returned a 64-bit value to the PAR, PAR.F==0:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>ATTR</td>
</tr>
<tr>
<td>61</td>
<td>PA</td>
</tr>
<tr>
<td>60</td>
<td>NS</td>
</tr>
<tr>
<td>59</td>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12</td>
</tr>
<tr>
<td>58</td>
<td>11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>
This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR can return a value that indicates the resulting attributes, rather than the values that appear in the translation table descriptors. More precisely:

- Memory attribute fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the translation table descriptors. This applies to the ATTR and SH fields.

**ATTR, bits [63:56]**

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in MAIR0 and MAIR1.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally UNKNOWN value.

**Bits [55:40]**

Reserved, RES0.

**PA, bits [39:12]**

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[39:12].

This field resets to an architecturally UNKNOWN value.

**LPAE, bit [11]**

When updating the PAR with the result of the translation operation, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Long-descriptor translation table format used. This means the PAR returned a 64-bit value.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**IMPLEMENTATION DEFINED, bit [10]**

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**NS, bit [9]**

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

This field resets to an architecturally UNKNOWN value.

**SH, bits [8:7]**

Shareability attribute, for the returned output address. Permitted values are:
### SH

<table>
<thead>
<tr>
<th>Meaning</th>
<th>SH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-shareable.</td>
<td>0b00</td>
</tr>
<tr>
<td>Outer Shareable.</td>
<td>0b10</td>
</tr>
<tr>
<td>Inner Shareable.</td>
<td>0b11</td>
</tr>
</tbody>
</table>

The value \(0b01\) is reserved.

---

**Note**

This field returns the value \(0b10\) for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

This field resets to an architecturally \texttt{UNKNOWN} value.

---

**Bits [6:1]**

Reserved, 

**F, bit \([0]\)**

Indicates whether the instruction performed a successful address translation.

\[\text{F} \quad \text{Meaning}\]

<table>
<thead>
<tr>
<th>Meaning</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address translation completed successfully.</td>
<td>0b0</td>
</tr>
</tbody>
</table>

This field resets to an architecturally \texttt{UNKNOWN} value.

---

### When the instruction returned a 64-bit value to the PAR, PAR.F==1:

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

**IMPLEMENTATION DEFINED, bits [63:56]**

**IMPLEMENTATION DEFINED.**

This field resets to an architecturally \texttt{UNKNOWN} value.

**IMPLEMENTATION DEFINED, bits [55:52]**

**IMPLEMENTATION DEFINED.**

This field resets to an architecturally \texttt{UNKNOWN} value.

**IMPLEMENTATION DEFINED, bits [51:48]**

**IMPLEMENTATION DEFINED.**

This field resets to an architecturally \texttt{UNKNOWN} value.
Bits [47:12]
Reserved, RES0.

LPAE, bit [11]
When updating the PAR with the result of the translation operation, this bit is set as follows:

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Long-descriptor translation table format used. This means the PAR returned a 64-bit value.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bit [10]
Reserved, RES0.

FSTAGE, bit [9]
Indicates the translation stage at which the translation aborted:

<table>
<thead>
<tr>
<th>FSTAGE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Translation aborted because of a fault in the stage 1 translation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Translation aborted because of a fault in the stage 2 translation.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

S2WLK, bit [8]
If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

This field resets to an architecturally UNKNOWN value.

Bit [7]
Reserved, RES0.

FST, bits [6:1]
Fault status field. Values are as in the DFSR.STATUS and IFSR.STATUS fields when using the Long-descriptor translation table format.

This field resets to an architecturally UNKNOWN value.

F, bit [0]
Indicates whether the instruction performed a successful address translation.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Address translation aborted.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Accessing the PAR
Accesses to this register use the following encodings:
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR_NS == '0' then
            return PAR_S<31:0>;
        else
            return PAR_NS<31:0>;
        end if
    else
        return PAR<31:0>;
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return PAR_NS<31:0>;
    else
        return PAR<31:0>;
    end if
elsif PSTATE.EL == EL3 then
    if SCR_NS == '0' then
        return PAR_S<31:0>;
    else
        return PAR_NS<31:0>;
    end if
end if

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR_NS == '0' then
            PAR_S = ZeroExtend(R[t]);
        else
            PAR_NS = ZeroExtend(R[t]);
        end if
    else
        PAR = ZeroExtend(R[t]);
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = ZeroExtend(R[t]);
    else
        PAR = ZeroExtend(R[t]);
    end if
elsif PSTATE.EL == EL3 then
    if SCR_NS == '0' then
        PAR_S = ZeroExtend(R[t]);
    else
        PAR_NS = ZeroExtend(R[t]);
    end if
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            PAR_S = R[t2]:R[t];
        else
            PAR_NS = R[t2]:R[t];
        end
    else
        return PAR;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = R[t2]:R[t];
    else
        PAR = R[t2]:R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S = R[t2]:R[t];
    else
        PAR_NS = R[t2]:R[t];
    end

MCRR{<coproc>, {#<opc1>, <Rt>, <Rt2>, <CRm>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0111</td>
<td>0b0000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            PAR_S = R[t2]:R[t];
        else
            PAR_NS = R[t2]:R[t];
    else
        return PAR;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = R[t2]:R[t];
    else
        PAR = R[t2]:R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S = R[t2]:R[t];
    else
        PAR_NS = R[t2]:R[t];
    end
PMCCFILTR, Performance Monitors Cycle Count Filter Register

The PMCCFILTR characteristics are:

**Purpose**

Determines the modes in which the Cycle Counter, PMCCNTR, increments.

**Configuration**

AArch32 System register PMCCFILTR bits [31:0] are architecturally mapped to AArch64 System register PMCCFILTR_EL0[31:0].

AArch32 System register PMCCFILTR bits [31:0] are architecturally mapped to External register PMCCFILTR_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCCFILTR are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCCFILTR is a 32-bit register.

**Field descriptions**

The PMCCFILTR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>U</td>
<td>NSK</td>
<td>NSU</td>
<td>NSH</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**P, bit [31]**

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMCCFILTR.NSK bit.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count cycles in EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in EL1.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**U, bit [30]**

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMCCFILTR.NSU bit.

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count cycles in EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**NSK, bit [29]**
When HaveEL(EL3):

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.
If the value of this bit is equal to the value of PMCCFILTR.P, cycles in Non-secure EL1 are counted.
Otherwise, cycles in Non-secure EL1 are not counted.
On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When HaveEL(EL3):

Non-secure EL0 (Unprivileged) filtering. Controls counting in Non-secure EL0.
If the value of this bit is equal to the value of PMCCFILTR.U, cycles in Non-secure EL0 are counted.
Otherwise, cycles in Non-secure EL0 are not counted.
On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When HaveEL(EL2):

Non-secure EL2 (Hyp mode) filtering bit. Controls counting in Non-secure EL2.

<table>
<thead>
<tr>
<th>NSH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not count cycles in EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count cycles in EL2.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [26:0]

Reserved, RES0.

Accessing the PMCCFILTR

PMCCFILTR can also be accessed by using PMXEVTYPER with PMSELR SEL set to 0b11111.
Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        end if;
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        end if;
    end if;
end if;
elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGETen == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
    return PMCCFILTR;
end if;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCFILTR;
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCFILTR;
    end if;
elsif PSTATE.EL == EL3 then
    return PMCCFILTR;
end if;

PMCCFILTR, Performance Monitors Cycle Count Filter Register

MCR{<c>{<q>}} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) & & PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() & & PMUSERENR.EN == '0' then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() & & ELUsingAArch32(EL2) & & HCR.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(0x00);
    else
      AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() & & !ELUsingAArch32(EL1) & & HCR_EL2.<E2H,TGE> != '11' & & (!HaveEL(EL3) ||
    SCR_EL3.FGTEn == '1') & & HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & & !ELUsingAArch32(EL2) & & MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & & ELUsingAArch32(EL2) & & HDCR.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(0x03);
  elsif HaveEL(EL3) & & !ELUsingAArch32(EL3) & & MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMCCFILTR = R[t];
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() & & !ELUsingAArch32(EL2) & & MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() & & ELUsingAArch32(EL2) & & HDCR.TPM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) & & !ELUsingAArch32(EL3) & & MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      PMCCFILTR = R[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) & & !ELUsingAArch32(EL3) & & MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      PMCCFILTR = R[t];
  elsif PSTATE.EL == EL3 then
    PMCCFILTR = R[t];
The PMCCNTR characteristics are:

**Purpose**

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for more information.

PMCCFILTR determines the modes and states in which the PMCCNTR can increment.

**Configuration**

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to AArch64 System register PMCCNTR_EL0[63:0].

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to External register PMCCNTR_EL0[63:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCCNTR are UNDEFINED.

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR continues to increment when clocks are stopped by WFI and WFE instructions.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCCNTR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

**Field descriptions**

The PMCCNTR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| CCNT |
| CCNT |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**CCNT, bits [63:0]**

Cycle count. Depending on the values of PMCR.\{LC,D\}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR.C sets this field to 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMCCNTR**

Accesses to this register use the following encodings:
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        end
    elsif ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        end
    end
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && E2H.TGE != '1' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
    end
elsif PSTATE.EL == EL3 then
    return PMCCNTR<31:0>;

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<<E2H,TGE> != '11' && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL2.FGTEn == '1') && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    PMCCNTR = ZeroExtend(R[t]);
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      PMCCNTR = ZeroExtend(R[t]);
else
  if PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      PMCCNTR = ZeroExtend(R[t]);
else
  if PSTATE.EL == EL3 then
  PMCCNTR = ZeroExtend(R[t]);

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1001</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
    end
elsif PSTATE.EL == EL3 then
    return PMCCNTR;
end

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b1001</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        endif
    elseif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        endif
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
    endif
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
    endif
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
    endif
elseif PSTATE.EL == EL3 then
    PMCCNTR = R[t2]:R[t];
The PMCEID0 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x0000 to 0x001F

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

For more information about the common events and the use of the PMCEIDn registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

AArch32 System register PMCEID0 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[31:0].

AArch32 System register PMCEID0 bits [31:0] are architecturally mapped to External register PMCEID0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCEID0 are UNDEFINED.

**Attributes**

PMCEID0 is a 32-bit register.

**Field descriptions**

The PMCEID0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

ID<\(n\)>, bit \([n]\), for \(n = 0 \) to \(31\)

ID\([n]\) corresponds to common event \(n\).

For each bit:

<table>
<thead>
<tr>
<th>ID&lt;(n)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**
Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

**Accessing the PMCEID0**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#\}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b110</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCEID0;
else
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return PMCEID0;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return PMCEID0;
else
  elsif PSTATE.EL == EL3 then
    return PMCEID0;
```
PMCEID1, Performance Monitors Common Event Identification register 1

The PMCEID1 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x0020 to 0x003F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

---

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

---

For more information about the common events and the use of the PMCEIDn registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

AArch32 System register PMCEID1 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[31:0].

AArch32 System register PMCEID1 bits [31:0] are architecturally mapped to External register PMCEID1[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCEID1 are UNDEFINED.

**Attributes**

PMCEID1 is a 32-bit register.

**Field descriptions**

The PMCEID1 bit assignments are:

<table>
<thead>
<tr>
<th>ID31</th>
<th>ID30</th>
<th>ID29</th>
<th>ID28</th>
<th>ID27</th>
<th>ID26</th>
<th>ID25</th>
<th>ID24</th>
<th>ID23</th>
<th>ID22</th>
<th>ID21</th>
<th>ID20</th>
<th>ID19</th>
<th>ID18</th>
<th>ID17</th>
<th>ID16</th>
<th>ID15</th>
<th>ID14</th>
<th>ID13</th>
<th>ID12</th>
<th>ID11</th>
<th>ID10</th>
<th>ID9</th>
<th>ID8</th>
</tr>
</thead>
</table>

ID<n>, bit [n], for n = 0 to 31

ID[n] corresponds to common event (0x0020 + n).

For each bit:

<table>
<thead>
<tr>
<th>ID&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

---

**Note**
Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID registers of that earlier version of the PMU architecture.

**Accessing the PMCEID1**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b111</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    end
  end
elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  end
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
  AArch32.TakeHypTrapException(0x00);
elsif EL2Enabled() && HCR.TGE == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PMCEID1 then
  AArch64.AArch32SystemAccessTrap(EL1, 0x03);
else
  return PMCEID1;
end
```

```c
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && HSTR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCEID1;
  end
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    return PMCEID1;
  end
elsif PSTATE.EL == EL3 then
  return PMCEID1;
```

PMCEID1, Performance Monitors Common Event Identification register 1
PMCEID2, Performance Monitors Common Event Identification register 2

The PMCEID2 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x4000 to 0x401F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

For more information about the common events and the use of the PMCEIDn registers see The section describing ‘Event numbers and common events’ in chapter D5 ‘The Performance Monitors Extension’ of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

AArch32 System register PMCEID2 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[63:32].

AArch32 System register PMCEID2 bits [31:0] are architecturally mapped to External register PMCEID2[63:32].

This register is present only when ARMv8.1-PMU is implemented. Otherwise, direct accesses to PMCEID2 are UNDEFINED.

**Attributes**

PMCEID2 is a 32-bit register.

**Field descriptions**

The PMCEID2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| \( \text{IDhi}^{\text{n}}, \text{bit} [n] \), for \( n = 0 \text{ to } 31 \) |

**\( \text{IDhi}^{\text{n}}, \text{bit} [n] \), for \( n = 0 \text{ to } 31 \)**

\( \text{IDhi}[n] \) corresponds to common event (0x4000 + n).

For each bit:

<table>
<thead>
<tr>
<th>( \text{\text{IDhi}^{\text{n}}} )</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0b0 )</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>( 0b1 )</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**
Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID\(<n>\) registers of that earlier version of the PMU architecture.

### Accessing the PMCEID2

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>}}\text{ <coproc>, \{<opc1>\}, <Rt>, <CRn}, \text{ <CRm}{, \{<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if AElEnabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03); 
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
    if AElEnabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03); 
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCEID2;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCEID2;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCEID2;
elsif PSTATE.EL == EL3 then
  return PMCEID2;
The PMCEID3 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x4020 to 0x403F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

For more information about the common events and the use of the PMCEIDn registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

AArch32 System register PMCEID3 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[63:32].

AArch32 System register PMCEID3 bits [31:0] are architecturally mapped to External register PMCEID3[63:32].

This register is present only when ARMv8.1-PMU is implemented. Otherwise, direct accesses to PMCEID3 are UNDEFINED.

**Attributes**

PMCEID3 is a 32-bit register.

**Field descriptions**

The PMCEID3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| IDhi<n>, bit [n], for n = 0 to 31 |

**IDhi<n>, bit [n], for n = 0 to 31**

IDhi[n] corresponds to common event (0x4020 + n).

For each bit:

<table>
<thead>
<tr>
<th>IDhi&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**
Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

**Accessing the PMCEID3**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc}1>, <\text{Rt}, <\text{CRn}, <\text{CRm}>\}, \{#<\text{opc}2>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b101</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if AElEnabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif PMUSERENR.EL0.EN == '0' then
        if AElEnabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch32.TakeHypTrapException(0x00);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID3;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMCEID3;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMCEID3;
    elsif PSTATE.EL == EL3 then
        return PMCEID3;
```

PMCEID3, Performance Monitors Common Event Identification register 3
PMCNTENCLR, Performance Monitors Count Enable Clear register

The PMCNTENCLR characteristics are:

**Purpose**

Disables the Cycle Count Register, PMCCNTR, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

PMCNTENCLR is used in conjunction with the PMCNTENSET register.

**Configuration**

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[31:0].

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to External register PMCNTENCLR_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCNTENCLR are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCNTENCLR is a 32-bit register.

**Field descriptions**

The PMCNTENCLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  | P<n>, bit [n] |

**C, bit [31]**

PMCCNTR disable bit. Disables the cycle counter register. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter is enabled. When written, disables the cycle counter.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter disable bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR_HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR_N.
When read, means that PMEVCNTR<\text{n}> is disabled. When written, has no effect.

When read, means that PMEVCNTR<\text{n}> is enabled. When written, disables PMEVCNTR<\text{n}>.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

## Accessing the PMCNTENCLR

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q}> <coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>{, \{#<opc2}\}}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL1) & PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      return PMCNTENCLR;
  else
    if PSTATE.EL == EL1 then
      if !ELUsingAArch32(EL1) && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
      else
        return PMCNTENCLR;
    elsif PSTATE.EL == EL2 then
      if !ELUsingAArch32(EL1) && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      else
        return PMCNTENCLR;
    elsif PSTATE.EL == EL3 then
      return PMCNTENCLR;
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
    endif
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
    endif
elsif PSTATE.EL == EL3 then
    PMCNTENCLR = R[t];
The PMCNTENSET characteristics are:

**Purpose**

Enables the Cycle Count Register, PMCCNTR, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

PMCNTENSET is used in conjunction with the PMCNTENCLR register.

**Configuration**

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[31:0].

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to External register PMCNTENSET_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCNTENSET are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMCNTENSET is a 32-bit register.

**Field descriptions**

The PMCNTENSET bit assignments are:

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>P&lt;n&gt;, bit [n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

**C, bit [31]**

PMCCNTR enable bit. Enables the cycle counter register. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter is enabled. When written, enables the cycle counter.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter enable bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR.HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.
PMCNTENSET, Performance Monitors Count Enable Set register

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that <strong>PMEVCNTR&lt;n&gt;</strong> is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that <strong>PMEVCNTR&lt;n&gt;</strong> event counter is enabled. When written, enables <strong>PMEVCNTR&lt;n&gt;</strong>.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

### Accessing the PMCNTENSET

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}\{<q>\} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if **PSTATE.EL** == **EL0** then
  if !**ELUsingAArch32(EL1)** && **PMUSERENR_EL0.EN** == '0' then
    if **EL2Enabled()** && !**ELUsingAArch32(EL2)** && **HCR_EL2.TGE** == '1' then
      **AArch64.AArch32SystemAccessTrap(EL2, 0x03);**
    else
      **AArch64.AArch32SystemAccessTrap(EL1, 0x03);**
  elsif **EL2Enabled()** && **PMUSERENR.EN** == '0' then
    if **EL2Enabled()** && !**ELUsingAArch32(EL2)** && **HCR_EL2.TGE** == '1' then
      **AArch64.AArch32SystemAccessTrap(EL2, 0x03);**
    elsif **EL2Enabled()** && **ELUsingAArch32(EL2)** && **HCR.TGE** == '1' then
      **AArch32.TakeHypTrapException(0x00);**
  else
    **UNDEFINED;**
  elsif **EL2Enabled()** && !**ELUsingAArch32(EL2)** && **HCR_EL2.<E2H,TGE> != '11'** && **HSTR_EL2.T9** == '1' then
    **AArch64.AArch32SystemAccessTrap(EL2, 0x03);**
  elsif **EL2Enabled()** && **ELUsingAArch32(EL2)** && **HSTR.T9** == '1' then
    **AArch32.TakeHypTrapException(0x03);**
  elsif **EL2Enabled()** && !**ELUsingAArch32(EL2)** && **MDCR_EL2.TPM** == '1' then
    **AArch64.AArch32SystemAccessTrap(EL2, 0x03);**
  elsif **EL2Enabled()** && **ELUsingAArch32(EL2)** && **HDCR.TPM** == '1' then
    **AArch32.TakeHypTrapException(0x03);**
  elsif **HaveEL(EL3)** && !**ELUsingAArch32(EL3)** && **MDCR_EL3.TPM** == '1' then
    **AArch64.AArch32SystemAccessTrap(EL3, 0x03);**
  else
    return **PMCNTENSET;**
  elsif **PSTATE.EL** == **EL1** then
    if **EL2Enabled()** && !**ELUsingAArch32(EL2)** && **HSTR_EL2.T9** == '1' then
      **AArch64.AArch32SystemAccessTrap(EL2, 0x03);**
    elsif **EL2Enabled()** && **ELUsingAArch32(EL2)** && **HSTR.T9** == '1' then
      **AArch32.TakeHypTrapException(0x03);**
    elsif **EL2Enabled()** && !**ELUsingAArch32(EL2)** && **MDCR_EL2.TPM** == '1' then
      **AArch64.AArch32SystemAccessTrap(EL2, 0x03);**
    elsif **EL2Enabled()** && **ELUsingAArch32(EL2)** && **HDCR.TPM** == '1' then
      **AArch32.TakeHypTrapException(0x03);**
    elsif **HaveEL(EL3)** && !**ELUsingAArch32(EL3)** && **MDCR_EL3.TPM** == '1' then
      **AArch64.AArch32SystemAccessTrap(EL3, 0x03);**
    else
      return **PMCNTENSET;**
  elsif **PSTATE.EL** == **EL2** then
    if **HaveEL(EL3)** && !**ELUsingAArch32(EL3)** && **MDCR_EL3.TPM** == '1' then
      **AArch64.AArch32SystemAccessTrap(EL3, 0x03);**
    else
      return **PMCNTENSET;**
  elsif **PSTATE.EL** == **EL3** then
    return **PMCNTENSET;**
<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    endif
else ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.TakeHypTrapException(0x00);
    endif
elsif ELUsingAArch32(EL1) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
else
    UNDEFINED;
else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    endif
    if !ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.TakeHypTrapException(0x00);
    endif
elsif !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
else
    PMCNTENSET = R[t];
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        endif
    else
        ELUsingAArch32(EL1) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch32.TakeHypTrapException(0x00);
        endif
else
    HaveEE(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch33SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
    endif
else
    PMCNTENSET = R[t];
else
    if !ELUsingAArch32(EL3) & MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
    endif
else
    PMCNTENSET = R[t];
else
    PMCNTENSET = R[t];
```
PMCR, Performance Monitors Control Register

The PMCR characteristics are:

**Purpose**

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

**Configuration**

AArch32 System register PMCR bits [31:0] are architecturally mapped to AArch64 System register `PMCR_EL0[31:0]`.

AArch32 System register PMCR bits [7:0] are architecturally mapped to External register `PMCR_EL0[7:0]`.

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMCR are **UNDEFINED**.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

PMCR is a 32-bit register.

**Field descriptions**

The PMCR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMP</td>
<td>IDCODE</td>
<td>N</td>
<td>RES0</td>
<td>LP</td>
<td>LC</td>
<td>DP</td>
<td>X</td>
<td>D</td>
<td>C</td>
<td>P</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IMP, bits [31:24]**

Implementer code. This field has an **IMPLEMENTATION DEFINED** value.

If this field is zero, then PMCR.IDCODE is **RES0** and software must use `MIDR` to identify the PE.

Otherwise this field and PMCR.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as `MIDR` Implementer.

Access to this field is **RO**.

**IDCODE, bits [23:16]**

When PMCR.IMP != 0x00:

Identification code. This field has an **IMPLEMENTATION DEFINED** value.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

Access to this field is **RO**.

Otherwise:

Reserved, **RES0**.
N, bits [15:11]

Indicates the number of event counters implemented. This value is in the range of 0b000000-0b111111. If the value is 0b000000 then only PMCCNTR is implemented. If the value is 0b111111 PMCCNTR and 31 event counters are implemented.

In an implementation that includes EL2:

- If EL2 is using AArch32, reads of this field from Non-secure EL1 and Non-secure EL0 return the value of HDCR.HPMN.
- If EL2 is using AArch64 and enabled in the current Security state, reads of this field from EL1 and EL0 return the value of MDCR_EL2.HPMN.

Access to this field is RO.

Bits [10:8]

Reserved, RES0.

LP, bit [7]

When ARMv8.5-PMU is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

<table>
<thead>
<tr>
<th>LP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR&lt;n&gt;[31:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR&lt;n&gt;[63:0].</td>
</tr>
</tbody>
</table>

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is RW or RAZ/WI.

If EL2 is implemented and HDCR.HPMN or MDCR_EL2.HPMN is less than PMCR.N, this bit does not affect the operation of event counters in the range [HDCR.HPMN..(PMCR.N-1)] or [MDCR_EL2.HPMN..(PMCR.N-1)].

PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.

Note

The effect of HDCR.HPMN or MDCR_EL2.HPMN on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, and regardless of whether EL2 is enabled in the current Security state. For more information, see the description of HDCR.HPMN or MDCR_EL2.HPMN.

On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

LC, bit [6]

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

<table>
<thead>
<tr>
<th>LC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR[31:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR[63:0].</td>
</tr>
</tbody>
</table>
Arm deprecates use of PMCR.LC = 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**DP, bit [5]**

*When HaveEL(EL3) or (ARMv8.1-PMU is implemented and HaveEL(EL2)):*

Disable cycle counter when event counting is prohibited.

<table>
<thead>
<tr>
<th>DP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting by PMCCNTR is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>When event counting for counters in the range [0..(HDCR.HPMN-1)] or [0..(MDCR_EL2.HPMN-1)] is prohibited, cycle counting by PMCCNTR is disabled.</td>
</tr>
</tbody>
</table>

For more information about the interaction between the Performance Monitors and EL3, see 'Effect of EL3 and EL2’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile

On a Warm reset, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**X, bit [4]**

*When the implementation includes an PMU event export bus:*

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

<table>
<thead>
<tr>
<th>X</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not export events.</td>
</tr>
<tr>
<td>0b1</td>
<td>Export events where not prohibited.</td>
</tr>
</tbody>
</table>

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device, for example to an OPTIONAL PE trace unit.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

On a Warm reset, this field resets to 0.

**Otherwise:**

Reserved, RAZ/WI.

**D, bit [3]**

Clock divider. The possible values of this bit are:

<table>
<thead>
<tr>
<th>D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When enabled, PMCCNTR counts every clock cycle.</td>
</tr>
<tr>
<td>0b1</td>
<td>When enabled, PMCCNTR counts once every 64 clock cycles.</td>
</tr>
</tbody>
</table>

If PMCR.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR.D = 1.

On a Warm reset, this field resets to 0.
C, bit [2]

Cycle counter reset. The effects of writing to this bit are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset PMCCNTR to zero.</td>
</tr>
</tbody>
</table>

This bit is always RAZ.

**Note**

Resetting PMCCNTR does not change the cycle counter overflow bit.

The value of PMCR_EL0.LC is ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is WO.

P, bit [1]

Event counter reset. The effects of writing to this bit are:

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset all event counters accessible in the current Exception level, not including PMCCNTR, to zero.</td>
</tr>
</tbody>
</table>

This bit is always RAZ.

In EL0 and EL1:

- If EL2 is implemented and enabled in the current Security state, and HDCR.HPMN or MDCR_EL2.HPMN is less than PMCR_EL0.N, a write of 1 to this bit does not reset event counters in the range [HDCR.HPMN..(PMCR.N-1)] or [MDCR_EL2.HPMN..(PMCR.N-1)].
- If EL2 is not implemented, EL2 is disabled in the current Security state, or HDCR.HPMN or MDCR_EL2.HPMN is equal to PMCR_EL0.N, a write of 1 to this bit resets all the event counters.

In EL2 and EL3, a write of 1 to this bit resets all the event counters.

**Note**

Resetting the event counters does not change the event counter overflow bits.

If ARMv8.5-PMU is implemented, the values of HDCR.HLP and PMCR.LP are ignored and bits [63:0] of all affected event counters are reset.

Access to this field is WO.

E, bit [0]

Enable.

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All event counters in the range [0..(PMN-1)] and PMCCNTR, are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>All event counters in the range [0..(PMN-1)] and PMCCNTR, are enabled by PMCNTENSET.</td>
</tr>
</tbody>
</table>

If EL2 is implemented then:

- If EL2 is using AArch32, PMN is HDCR.HPMN.
- If EL2 is using AArch64, PMN is MDCR_EL2.HPMN.
- If PMN is less than PMCR.N, this bit does not affect the operation of event counters in the range [PMN..(PMCR.N-1)].
If EL2 is not implemented, PMN is PMCR.N.

**Note**

The effect of MDCR_EL2.HPMN or HDCR.HPMN on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDCR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to 0.

**Accessing the PMCR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch32.TakeHypTrapException(0x00);
      UNDEFINED;
  endif
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCR;
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    return PMCR;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMCR;
  endif
elsif PSTATE.EL == EL3 then
  return PMCR;
endif

MCR{<c>{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArm32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArm32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elseif ELUsingAArm32(EL1) && PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArm32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArm32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elseif EL2Enabled() && !ELUsingAArm32(EL2) && HCR_EL2.TGE != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArm32(EL2) && HSTR.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && !ELUsingAArm32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
 elseif EL2Enabled() && !ELUsingAArm32(EL2) && MDCR_EL2.TPMCR == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArm32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && ELUsingAArm32(EL2) && HDCR.TPMCR == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArm32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMCR = R[t];
  elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArm32(EL2) && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArm32(EL2) && HSTR.T9 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArm32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArm32(EL2) && MDCR_EL2.TPMCR == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArm32(EL2) && HDCR.TPM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && ELUsingAArm32(EL2) && HDCR.TPMCR == '1' then
      AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArm32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      PMCR = R[t];
  elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArm32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      PMCR = R[t];
  else
    PMCR = R[t];
PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n> characteristics are:

**Purpose**

Holds event counter n, which counts events, where n is 0 to 30.

**Configuration**

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to AArch64 System register PMEVCNTR<n>_EL0[31:0].

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to External register PMEVCNTR<n>_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMEVCNTR<n> are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMEVCNTR<n> is a 32-bit register.

**Field descriptions**

The PMEVCNTR<n> bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Event counter n</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If ARMv8.5-PMU is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMEVCNTR<n> return bits [31:0] of the counter.
- Writes to PMEVCNTR<n> update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64 at any Exception level, bits [63:32] are not required to be implemented.

If ARMv8.5-PMU is not implemented, the event counter is 32 bits. On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMEVCNTR<n>**

PMEVCNTR<n> can also be accessed by using PMXEVCNTR with PMSELR.SEL set to the value of <n>.

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, and EL1 is using AArch64, for permitted reads and writes at EL0:
If \(<n>\) is an unimplemented event counter, the access is UNDEFINED.
Otherwise, if \(<n>\) is greater than or equal to \(\text{MDCR}_{\text{EL2}}.\text{HPMN}\), the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:

If \(<n>\) is greater than or equal to the number of accessible counters, reads and writes of \(\text{PMEVCNTR}<n>\) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- If EL2 is implemented and enabled in the current Security state, and \(<n>\) is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by \(\text{PMUSERENR}\).\{ER,EN\} or \(\text{PMUSERENR}_{\text{EL0}}.\{ER,EN\}\).

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, \(\text{HDCR}.\text{HPMN}\) identifies the number of accessible counters.
- If EL2 is using AArch64, \(\text{MDCR}_{\text{EL2}}.\text{HPMN}\) identifies the number of accessible counters.

Otherwise, the number of accessible counters is the number of implemented counters. See \(\text{HDCR}.\text{HPMN}\) and \(\text{MDCR}_{\text{EL2}}.\text{HPMN}\) for more details.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \text{<coproc>}, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b10:n[4:3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL1) && SCR_EL3.FGTEn == '1' && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
  if PSTATE.EL == EL1 then
    if !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
  elsif PSTATE.EL == EL3 then
    return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0000</td>
<td>0b1110</td>
<td>0b10:n[4:3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elseif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elseif EL2Enabled() && !ELUsingAArch32(EL1) && HDFGWTR_EL2.PMEVCNTrn_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && HDFGWTR_EL3.PMEVCNTrn_EL3 == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
  endif
elsif PSTATE.EL == EL3 then
  PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
endif
The PMEVTYPER\(n\) characteristics are:

**Purpose**

Configures event counter \(n\), where \(n\) is 0 to 30.

**Configuration**

AArch32 System register PMEVTYPER\(n\) bits [31:0] are architecturally mapped to AArch64 System register PMEVTYPER\(n\).EL0[31:0].

AArch32 System register PMEVTYPER\(n\) bits [31:0] are architecturally mapped to External register PMEVTYPER\(n\).EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMEVTYPER\(n\) are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMEVTYPER\(n\) is a 32-bit register.

**Field descriptions**

The PMEVTYPER\(n\) bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| P   | U   | NSK | NSU | NSH | RES0 | MT  | RES0 | evtCount[15:10] | evtCount[9:0] |

**P, bit [31]**

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMEVTYPER\(n\).NSK bit.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events in EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in EL1.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**U, bit [30]**

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMEVTYPER\(n\).NSU bit.

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events in EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**NSK, bit [29]**
When HaveEL(EL3):

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of PMEVTYPER<n>.P, events in Non-secure EL1 are counted.

Otherwise, events in Non-secure EL1 are not counted.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**NSU, bit [28]**

When HaveEL(EL3):

Non-secure EL0 (Unprivileged) filtering. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of PMEVTYPER<n>.U, events in Non-secure EL0 are counted.

Otherwise, events in Non-secure EL0 are not counted.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**NSH, bit [27]**

When HaveEL(EL2):

Non-secure EL2 (Hyp mode) filtering bit. Controls counting in Non-secure EL2.

<table>
<thead>
<tr>
<th>NSH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not count events in EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count events in EL2.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bit [26]**

Reserved, RES0.

**MT, bit [25]**

When ARMv8.6-MTPMU is implemented or IMPLEMENTATION DEFINED multi-threaded PMU Extension is implemented:

Multithreading.

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events only on controlling PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count events from any PE with the same affinity at level 1 and above as this PE.</td>
</tr>
</tbody>
</table>
When the lowest level of affinity consists of logical PEs that are implemented using a multi-threading type approach, an implementation is described as multi-threaded. That is, the performance of PEs at the lowest affinity level is highly interdependent.

Events from a different thread of a multithreaded implementation are not attributable to the thread counting the event.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [24:16]**

Reserved, RES0.

evtCount[15:10], bits [15:10]

*When ARMv8.1-PMU is implemented:*

Extension to evtCount[9:0]. See evtCount[9:0] for more details.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count. The event number of the event that is counted by event counter PMEVCNTR<n>. Software must program this field with an event that is supported by the PE being programmed.

The ranges of event numbers allocated to each type of event are shown in Allocation of the PMU event number space.

If evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- If 16-bit evtCount is implemented, for the range 0x4000 to 0x403F, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED events, it is UNPREDICTABLE what event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

**Note**

UNPREDICTABLE means the event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include an event from a set of common IMPLEMENTATION DEFINED events, then no event is counted and the value read back on evtCount is the value written.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMEVTYPER<n>**

PMEVTYPER<n> can also be accessed by using PMXEVTYPER with PMSELR SEL set to n.
If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, and EL1 is using AArch64, for permitted reads and writes at EL0:

- If \( n \) is an unimplemented event counter, the access is **UNDEFINED**.
- Otherwise, if \( n \) is greater than or equal to \( \text{MDCR\_EL2}.\text{HPMN} \), the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:

If \( n \) is greater or equal to the number of accessible counters, reads and writes of \( \text{PMEVTYPER}\_n \) are **CONSTRAINED UNPREDICTABLE**, and the following behaviors are permitted:

- Accesses to the register are **UNDEFINED**.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a **NOP**.
- If EL2 is implemented and enabled in the current Security state, and \( n \) is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

**Note**

In EL0, an access is permitted if it is enabled by \( \text{PMUSERENR}.\text{EN} \) or \( \text{PMUSERENR}\_\text{EL0}.\text{EN} \).

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, \( \text{HDCR}.\text{HPMN} \) identifies the number of accessible counters.
- If EL2 is using AArch64, \( \text{MDCR\_EL2}.\text{HPMN} \) identifies the number of accessible counters.

Otherwise, the number of accessible counters is the number of implemented counters. See \( \text{HDCR}.\text{HPMN} \) and \( \text{MDCR\_EL2}.\text{HPMN} \) for more details.

Accesses to this register use the following encodings:

\[
\text{MRC}\{c\}\{q\} \langle\text{coproc}\rangle, \{\#\}\langle\text{opc1}\rangle, \langle\text{Rt}\rangle, \langle\text{CRn}\rangle, \langle\text{CRm}\rangle\{, \{\#\}\langle\text{opc2}\rangle\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b11:n[4:3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
    elseif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
    elseif PSTATE.EL == EL3 then
        return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
end

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b11:n[4:3]</td>
<td>n[2:0]</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    else
        if ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                UNDEFINED;
        end else
        if EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
        end else
        if EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
                    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
                else
                    PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
                end else
            end else
        end else
    end else
end if
else
    if PSTATE.EL == EL1 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
        end else
    end if
else
    if PSTATE.EL == EL3 then
        PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
    else
        PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
    end if
end if
PMINTENCLR, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR characteristics are:

**Purpose**

Disables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR, and the event counters PMEVCNTR<n>. Reading the register shows which overflow interrupt requests are enabled.

PMINTENCLR is used in conjunction with the PMINTENSET register.

**Configuration**

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0].

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to External register PMINTENCLR_EL1[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMINTENCLR are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMINTENCLR is a 32-bit register.

**Field descriptions**

The PMINTENCLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>C</td>
<td>PMCCNTR overflow interrupt request disable bit. Possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow interrupt request disable bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR.HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.
PMINTENCLR, Performance Monitors Interrupt Enable Clear register

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that the PMEVCNTR&lt;n&gt; event counter interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that the PMEVCNTR&lt;n&gt; event counter interrupt request is enabled. When written, disables the PMEVCNTR&lt;n&gt; interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMINTENCLR**

Accesses to this register use the following encodings:

MRC{c}{q} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMINTENCLR;
    endif
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMINTENCLR;
    endif
elsif PSTATE.EL == EL3 then
    return PMINTENCLR;
endif

MCR{c}{q} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENCLR = R[t];
    end if;
elsif PSTATE.EL == EL2 then
    if EL2Enabled() && ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENCLR = R[t];
    end if;
elsif PSTATE.EL == EL3 then
    PMINTENCLR = R[t];
PMINTENSET, Performance Monitors Interrupt Enable Set register

The PMINTENSET characteristics are:

**Purpose**

Enables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR, and the event counters PMEVCNTR<n>. Reading the register shows which overflow interrupt requests are enabled.

PMINTENSET is used in conjunction with the PMINTENCLR register.

**Configuration**

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to AArch64 System register PMINTENSET_EL1[31:0].

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to External register PMINTENSET_EL1[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMINTENSET are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMINTENSET is a 32-bit register.

**Field descriptions**

The PMINTENSET bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  | P<n>, bit [n] |

**C, bit [31]**

PMCCNTR overflow interrupt request enable bit. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter overflow interrupt request is</td>
</tr>
<tr>
<td></td>
<td>disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter overflow interrupt request is</td>
</tr>
<tr>
<td></td>
<td>enabled. When written, enables the cycle count overflow interrupt</td>
</tr>
<tr>
<td></td>
<td>request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow interrupt request enable bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR.HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.
PMINTENSET, Performance Monitors Interrupt Enable Set register

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that the PMEVCTR&lt;n&gt; event counter interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that the PMEVCTR&lt;n&gt; event counter interrupt request is enabled. When written, enables the PMEVCTR&lt;n&gt; interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMINTENSET**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>\{, \{#\}<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMINTENSET;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMINTENSET;
elsif PSTATE.EL == EL3 then
  return PMINTENSET;

MRC\{<c>\}{<q>} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm>\{, \{#\}<opc2>\}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
eslf PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMINTENSET = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMINTENSET = R[t];
    elsif PSTATE.EL == EL3 then
        PMINTENSET = R[t];
PMMIR, Performance Monitors Machine Identification Register

The PMMIR characteristics are:

**Purpose**

Describes Performance Monitors parameters specific to the implementation to software.

**Configuration**

This register is present only when ARMv8.4-PMU is implemented. Otherwise, direct accesses to PMMIR are UNDEFINED.

**Attributes**

PMMIR is a 32-bit register.

**Field descriptions**

The PMMIR bit assignments are:

```
   31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
   RES0  SLOTS
```

**Bits [31:8]**

Reserved, RES0.

**SLOTS, bits [7:0]**

Operation width. The largest value by which the STALL_SLOT event might increment by in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

**Accessing the PMMIR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMMIR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMMIR;
        end elsif
    elsif PSTATE.EL == EL3 then
        return PMMIR;
end if
PMOVSR, Performance Monitors Overflow Flag Status Register

The PMOVSR characteristics are:

**Purpose**

Contains the state of the overflow bit for the Cycle Count Register, \texttt{PMCCNTR}, and each of the implemented event counters \texttt{PMEVCNTR<n>}. Writing to this register clears these bits.

**Configuration**

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to AArch64 System register \texttt{PMOVSCLR_EL0[31:0]}.

AArch32 System register PMOVSR bits [31:0] are architecturally mapped to External register \texttt{PMOVSCLR_EL0[31:0]}.

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMOVSR are \texttt{UNDEFINED}.

RW fields in this register reset to architecturally \texttt{UNKNOWN} values.

**Attributes**

PMOVSR is a 32-bit register.

**Field descriptions**

The PMOVSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  | P<n>, bit [n] |

**C, bit [31]**

Cycle counter overflow clear bit. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter has not overflowed since this bit</td>
</tr>
<tr>
<td></td>
<td>was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter has overflowed since this bit</td>
</tr>
<tr>
<td></td>
<td>was last cleared. When written, clears the cycle counter overflow</td>
</tr>
<tr>
<td></td>
<td>bit to 0.</td>
</tr>
</tbody>
</table>

\texttt{PMCR.LC} controls whether an overflow is detected from unsigned overflow of \texttt{PMCCNTR[31:0]} or unsigned overflow of \texttt{PMCCNTR[63:0]}.

On a Warm reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow clear bit for \texttt{PMEVCNTR<n>}.

If \(N\) is less than 31, then bits [30:3] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, \(N\) is the value in \texttt{MDCR_EL2.HPMN} if EL2 is using AArch64, or in \texttt{HDCR.HPMN} if EL2 is using AArch32. Otherwise, \(N\) is the value in \texttt{PMCR.N}.
<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt; has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt; has overflowed since this bit was last cleared. When written, clears the PMEVCNTR&lt;n&gt; overflow bit to 0.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, MDCR_EL2.HLP, HDCR.HLP, and PMCR LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<n>[31:0] or unsigned overflow of PMEVCNTR<n>[63:0]. PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMOVSR**

Accesses to this register use the following encodings:

\[ \text{MRC\{<c>\}{<q> \coproc\}, \{#<opc1>, \Rt, \CRn, \CRm\}, \{#<opc2>\}} \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        endelse
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSR;
    endelse
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSR;
    endelse
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSR;
    endelse
elsif PSTATE.EL == EL3 then
    return PMOVSR;
endelse

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    if EL2Enabled() && !ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      else
        UNDEFINED;
    else
      if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
      else
        PMOVSR = R[t];
  end if
else
  if PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      if EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
      elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
      elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
      elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
      else
        PMOVSR = R[t];
  end if
elsif PSTATE.EL == EL2 then
  if EL2Enabled() && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMOVSR = R[t];
elsif PSTATE.EL == EL3 then
  PMOVSR = R[t];
PMOVSSET, Performance Monitors Overflow Flag Status Set register

The PMOVSSET characteristics are:

**Purpose**

Sets the state of the overflow bit for the Cycle Count Register, PMCCNTR, and each of the implemented event counters PMEVCNTR<n>.

**Configuration**

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[31:0].

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to External register PMOVSSET_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMOVSSET are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMOVSSET is a 32-bit register.

**Field descriptions**

The PMOVSSET bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P&lt;n&gt;, bit [n]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**C, bit [31]**

Cycle counter overflow set bit.

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter has overflowed since this bit was last cleared. When written, sets the cycle counter overflow bit to 1.</td>
</tr>
</tbody>
</table>

PMCR_LC controls whether an overflow is detected from unsigned overflow of PMCCNTR[31:0] or unsigned overflow of PMCCNTR[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow set bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR_HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.

Page 2895
<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt; has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt; has overflowed since this bit was last set. When written, sets the PMEVCNTR&lt;n&gt; overflow bit to 1.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, MDCR_EL2.HLP, HDCR.HLP, and PMCR_LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<n>[31:0] or unsigned overflow of PMEVCNTR<n>[63:0]. PMEVCNTR<n>[63:32] cannot be accessed directly in AArch32 state.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMOVSSET**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  endif
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
    endif
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMOVSSET;
  endif
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMOVSSET;
  endif
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMOVSSET;
  endif
elsif PSTATE.EL == EL3 then
  return PMOVSSET;
endif

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERREN_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERREN.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
        then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL3, 0x03);
    else
        PMOVSSET = R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL3, 0x03);
        else
            PMOVSSET = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemSystemAccessTrap(EL3, 0x03);
        else
            PMOVSSET = R[t];
    elsif PSTATE.EL == EL3 then
        PMOVSSET = R[t];
PMSELR, Performance Monitors Event Counter Selection Register

The PMSELR characteristics are:

**Purpose**

Selects the current event counter PMEVCNTR<\textit{n}> or the cycle counter, CCNT.

PMSELR is used in conjunction with PMXEVTYPER to determine the event that increments a selected event counter, and the modes and states in which the selected counter increments.

It is also used in conjunction with PMXEVCNTR, to determine the value of a selected event counter.

**Configuration**

AArch32 System register PMSELR bits [31:0] are architecturally mapped to AArch64 System register PMSELR_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMSELR are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMSELR is a 32-bit register.

**Field descriptions**

The PMSELR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [31:5]**

Reserved, RES0.

**SEL, bits [4:0]**

Selects event counter, PMEVCNTR<\textit{n}>, where \textit{n} is the value held in this field. This value identifies which event counter is accessed when a subsequent access to PMXEVTYPER or PMXEVCNTR occurs.

This field can take any value from 0 (0b00000) to (PMCR.N)-1, or 31 (0b11111).

When PMSELR.SEL is 0b11111, it selects the cycle counter and:

- A read of the PMXEVTYPER returns the value of PMCCFILTR.
- A write of the PMXEVTYPER writes to PMCCFILTR.
- A read or write of PMXEVCNTR has CONSTRAINED UNPREDICTABLE effects. See PMXEVCNTR for more details.

For details of the results of accesses to event counters, see PMXEVTYPER and PMXEVCNTR.

For information about the number of counters accessible at each Exception level, see HDCR.HPMN and MDCR_EL2.HPMN.

On a Warm reset, this field resets to an architecturally UNKNOWN value.
Accessing the PMSELR

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !HaveEL(EL3) || SCR_EL3.FGTEn == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HDFGTR_EL2.PMSELR_EL0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
else
  return PMSELR;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
else
  return PMSELR;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  return PMSELR;
elsif PSTATE.EL == EL3 then
  return PMSELR;
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMSELR = R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMSELR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMSELR = R[t];
    elsif PSTATE.EL == EL3 then
        PMSELR = R[t];

27/09/2019 18:22; 6134483bd14dc812a99e984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
PMSWINC, Performance Monitors Software Increment register

The PMSWINC characteristics are:

**Purpose**

Increments a counter that is configured to count the Software increment event, event 0x00. For more information, see ‘SW_INCR’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D5.

**Configuration**

AArch32 System register PMSWINC bits [31:0] are architecturally mapped to AArch64 System register PMSWINC_EL0[31:0].

AArch32 System register PMSWINC bits [31:0] are architecturally mapped to External register PMSWINC_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMSWINC are UNDEFINED.

**Attributes**

PMSWINC is a 32-bit register.

**Field descriptions**

The PMSWINC bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 | P<n>, bit [n] |

**Bit [31]**

Reserved, RES0.

**P<n>, bit [n], for n = 0 to 30**

Event counter software increment bit for PMEVCNTR<n>.

If N is less than 31, then bits [30:N] are WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in MDCR_EL2.HPMN if EL2 is using AArch64, or in HDCR.HPMN if EL2 is using AArch32. Otherwise, N is the value in PMCR.N.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action. The write to this bit is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>If PMEVCNTR&lt;n&gt; is enabled and configured to count the software increment event, increments PMEVCNTR&lt;n&gt; by 1. If PMEVCNTR&lt;n&gt; is disabled, or not configured to count the software increment event, the write to this bit is ignored.</td>
</tr>
</tbody>
</table>

**Accessing the PMSWINC**

Accesses to this register use the following encodings:
PMSWINC, Performance Monitors Software Increment register

if PSTATE.EL == EL0 then
    if !ELUSingAArch32(EL1) && PMUSERENR_EL0.<SW,EN> == '00' then
        if EL2Enabled() && !ELUSingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUSingAArch32(EL1) && PMUSERENR.<SW,EN> == '00' then
        if EL2Enabled() && !ELUSingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUSingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUSingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUSingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        PMSWINC = R[t];
    endif
else
    if PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUSingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            PMSWINC = R[t];
        endif
    elseif PSTATE.EL == EL2 then
        if EL2Enabled() && !ELUSingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            PMSWINC = R[t];
        endif
    elseif PSTATE.EL == EL3 then
        PMSWINC = R[t];
    endif
PMUSERENR, Performance Monitors User Enable Register

The PMUSERENR characteristics are:

**Purpose**

Enables or disables User mode access to the Performance Monitors.

**Configuration**

AArch32 System register PMUSERENR bits [31:0] are architecturally mapped to AArch64 System register PMUSERENR_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMUSERENR are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PMUSERENR is a 32-bit register.

**Field descriptions**

The PMUSERENR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, reserved</td>
</tr>
<tr>
<td>3-29</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>ER</td>
<td>Event counter read trap control</td>
</tr>
<tr>
<td>2</td>
<td>CR</td>
<td>Cycle counter read trap control</td>
</tr>
<tr>
<td>1-0</td>
<td>Reserved</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**ER, bit [3]**

Event counter read trap control:

<table>
<thead>
<tr>
<th>ER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 reads of the PMXEVCNTR and PMEVCNTR&lt;n&gt;, and EL0 RW access to the PMSELR, are trapped to Undefined mode if PMUSERENR.EN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overrides PMUSERENR.EN and enables RO access to PMXEVCNTR and PMEVCNTR&lt;n&gt;, and RW access to PMSELR.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**CR, bit [2]**

Cycle counter read trap control:

<table>
<thead>
<tr>
<th>CR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 reads of the PMCCNTR are trapped to Undefined mode if PMUSERENR.EN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overrides PMUSERENR.EN and enables access to PMCCNTR.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.
**SW, bit [1]**

Software increment write trap control:

<table>
<thead>
<tr>
<th>SW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 writes to the PMSWINC are trapped to Undefined mode if PMUSERENR.EN is also 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overrides PMUSERENR.EN and enables access to PMSWINC.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**EN, bit [0]**

Traps EL0 accesses to the Performance Monitors registers to Undefined mode:

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>While at EL0, PMUSERENR is always RO. Accesses to the other Performance Monitors registers are trapped to Undefined mode, unless enabled by one of PMUSERENR.{ER, CR, SW}.</td>
</tr>
<tr>
<td>0b1</td>
<td>While at EL0, software can access all PMU registers except PMINTENSET and PMINTENCLR.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**Accessing the PMUSERENR**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}\{<q>\} <coproc>, \{#\}<opc1>, <Rt>, <CRn>, <CRm\}, \{#\}<opc2}>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '1' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '1' && HSTR.09 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.09 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    return PMUSERENR;
  end if
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMUSERENR;
  end if
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMUSERENR = R[t ];
  end if
else
  PMUSERENR = R[t ];
endif

if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '1' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMUSERENR = R[t ];
  end if
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMUSERENR = R[t ];
  end if
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMUSERENR = R[t ];
  end if
elsif PSTATE.EL == EL3 then
  return PMUSERENR;
end if

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
PMXEVCNTR, Performance Monitors Selected Event Count Register

The PMXEVCNTR characteristics are:

**Purpose**

Reads or writes the value of the selected event counter, \texttt{PMEVCNTR<n>}. \texttt{PMSELR}.SEL determines which event counter is selected.

**Configuration**

AArch32 System register PMXEV CNTR bits [31:0] are architecturally mapped to AArch64 System register \texttt{PMXEV CNTR_EL0[31:0]}. This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMXEV CNTR are \texttt{UNDEFINED}. RW fields in this register reset to architecturally \texttt{UNKNOWN} values.

**Attributes**

PMXEV CNTR is a 32-bit register.

**Field descriptions**

The PMXEV CNTR bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMSELRSEL</td>
<td>Performance Monitors Selected Event Count Register</td>
</tr>
<tr>
<td>PMXEV CNTR</td>
<td>Value of the selected event counter, \texttt{PMEVCNTR&lt;n&gt;}, where n is the value stored in \texttt{PMSELR}.SEL.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMXEV CNTR return bits [31:0] of the counter.
- Writes to PMXEV CNTR update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64 at any Exception level, bits [63:32] are not required to be implemented.

If ARMv8.5-PMU is not implemented, the event counter is 32 bits.

On a Warm reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the PMXEV CNTR**

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, and EL1 is using AArch64, for permitted reads and writes at EL0:

- If \texttt{PMSELR}.SEL is an unimplemented event counter, the access is \texttt{UNDEFINED}.
- Otherwise, if \texttt{PMSELR}.SEL is greater than or equal to \texttt{MDCR_EL2}.HPMN, the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:
If PMSELR_SEL is greater than or equal to the number of accessible counters then reads and writes of PMXEVCNTR are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR_SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and PMSELR_SEL is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

**Note**

In EL0, an access is permitted if it is enabled by PMUSERENR_EL0, {ER,EN} or PMUSERENR_EL0, {ER,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR_HPMN identifies the number of accessible counters.
- If EL2 is using AArch64, MDCR_EL2_HPMN identifies the number of accessible counters.

Otherwise, the number of accessible counters is the number of implemented counters. See HDCR_HPMN and MDCR_EL2_HPMN for more details.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} \text{<coproc>}, \{#\}<\text{opc1}>, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{#\}<\text{opc2}>\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
    else
      UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    return PMXEVCNTR;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return PMXEVCNTR;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
      return PMXEVCNTR;
  elsif PSTATE.EL == EL3 then
    return PMXEVCNTR;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, {<Rt>}, {<CRn>}, {<CRm>}, {#}<opc2>

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch32.TakeHypTrapException(0x00);
  else
    UNDEFINED;
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMXEVCNTR = R[t];
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
      AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMXEVCNTR = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMXEVCNTR = R[t];
elsif PSTATE.EL == EL3 then
  PMXEVCNTR = R[t];
PMXEVTYPER, Performance Monitors Selected Event Type Register

The PMXEVTYPER characteristics are:

**Purpose**

When PMSELR SEL selects an event counter, this accesses a PMEVTYPER<n> register. When PMSELR_SEL selects the cycle counter, this accesses PMCCFILTR.

**Configuration**

AArch32 System register PMXEVTYPER bits [31:0] are architecturally mapped to AArch64 System register PMXEVTYPER_EL0[31:0].

This register is present only when PMUv3 is implemented. Otherwise, direct accesses to PMXEVTYPER are UNDEFINED.

**Attributes**

PMXEVTYPER is a 32-bit register.

**Field descriptions**

The PMXEVTYPER bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Event type register or PMCCFILTR**

**Bits [31:0]**

Event type register or PMCCFILTR.

When PMSELR_SEL == 31, this register accesses PMCCFILTR.

Otherwise, this register accesses PMEVTYPER<n> where n is the value in PMSELR_SEL.

**Accessing the PMXEVTYPER**

If ARMv8.6-FGT is implemented, EL2 is implemented and enabled in the current Security state, and EL1 is using AArch64, for permitted reads and writes at EL0:

- If PMSELR_SEL is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, if PMSELR_SEL is not 31, and is greater than or equal to MDCR_EL2.HPMN, the access is trapped to EL2.

If ARMv8.6-FGT is not implemented, or the above behaviors do not apply:

If PMSELR_SEL is not 31, and is greater than or equal to the number of accessible counters then reads and writes of PMXEVTYPER are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if PMSELR_SEL has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if PMSELR_SEL is 31.
If EL2 is implemented and enabled in the current Security state, and PMSELR.SEL is less than the number of implemented counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

**Note**

In EL0, an access is permitted if it is enabled by PMUSERENR.EN or PMUSERENR_EL0.EN.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, HDCR.HPMN identifies the number of accessible counters.
- If EL2 is using AArch64, MDCR_EL2.HPMN identifies the number of accessible counters.

Otherwise, the number of accessible counters is the number of implemented counters. See HDCR.HPMN and MDCR_EL2.HPMN for more details.

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>\{\text{(#}<opc1>, Rt>}, <CRn>, <CRm}\{\text{(#}<opc2>)}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return PMXEVTYPER;
    end if;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVTYPER;
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVTYPER;
    end if;
elsif PSTATE.EL == EL3 then
    return PMXEVTYPER;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  end if
else
  AArch64.AArch32SystemAccessTrap(EL1, 0x03);
 elsif ELUsingAArch32(EL1) && PMUSERENR.EN == '0' then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  end if
elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
  AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
  AArch32.TakeHypTrapException(0x03);
elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
  AArch64.AArch32SystemAccessTrap(EL3, 0x03);
else
  PMXEVTYPER = R[t];
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    AArch64.AArch32SystemAccessTrap(EL1, 0x03);
  end if
elsif PSTATE.EL == EL2 then
  if !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    PMXEVTYPER = R[t];
elsif PSTATE.EL == EL3 then
  PMXEVTYPER = R[t];
PRRR, Primary Region Remap Register

The PRRR characteristics are:

**Purpose**

Controls the top level mapping of the TEX[0], C, and B memory region attributes.

**Configuration**

AArch32 System register PRRR bits [31:0] are architecturally mapped to AArch64 System register MAIR_EL1[31:0] when TTBCR.EAE == 0.

MAIR0 and PRRR are the same register, with a different view depending on the value of TTBCR.EAE:

- When it is set to 0, the register is as described in PRRR.
- When it is set to 1, the register is as described in MAIR0.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

PRRR is a 32-bit register.

**Field descriptions**

The PRRR bit assignments are:

**When TTBCR.EAE == 0:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>12</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOS7</td>
<td>NOS6</td>
<td>NOS5</td>
<td>NOS4</td>
<td>NOS3</td>
<td>NOS2</td>
<td>NOS1</td>
<td>NOS0</td>
<td>RES0</td>
<td>NS1</td>
<td>NS0</td>
<td>DS1</td>
<td>DS0</td>
<td>TR7</td>
<td>TR6</td>
<td>TR5</td>
<td>TR4</td>
<td>TR3</td>
<td>TR2</td>
</tr>
</tbody>
</table>

**NOS<n>**, bit [n+24], for n = 0 to 7

Not Outer Shareable. NOS<n> is the Outer Shareable property for memory attributes n, if the region is mapped as Normal memory that is not Inner Non-cacheable, Outer Non-cacheable, and the appropriate PRRR.{NS0, NS1} field identifies the region as shareable. n is the value of the concatenation of the {TEX[0], C, B} bits from the translation table descriptor. The possible values of each NOS<n> field other than NOS6 are:

<table>
<thead>
<tr>
<th>NOS&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory region is Outer Shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory region is Inner Shareable.</td>
</tr>
</tbody>
</table>

The value of this bit is ignored if the region is:

- Device memory
- Normal memory that is at least one of:
  - Inner Non-cacheable, Outer Non-cacheable.
  - Identified by the appropriate PRRR.{NS0, NS1} field as Non-shareable.

The meaning of the NOS6 field is IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Bits [23:20]**

Reserved, RES0.
NS1, bit [19]

Mapping of S = 1 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the translation table descriptor set to 1.

The possible values of this bit are:

<table>
<thead>
<tr>
<th>NS1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Region is Non-shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Region is shareable. The value of the appropriate PRRR.NOS&lt;n&gt; field determines whether the region is Inner Shareable or Outer Shareable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

NS0, bit [18]

Mapping of S = 0 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the translation table descriptor set to 0.

The possible values of this bit are:

<table>
<thead>
<tr>
<th>NS0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Region is Non-shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Region is shareable. The value of the appropriate PRRR.NOS&lt;n&gt; field determines whether the region is Inner Shareable or Outer Shareable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

DS1, bit [17]

Mapping of S = 1 attribute for Device memory. From Armv8, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

This field resets to an architecturally UNKNOWN value.

DS0, bit [16]

Mapping of S = 0 attribute for Device memory. From Armv8, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

This field resets to an architecturally UNKNOWN value.

TR<n>, bits [2n+1:2n], for n = 0 to 7

TR<n> is the primary TEX mapping for memory attributes n, and defines the mapped memory type for a region with attributes n. n is the value of the concatenation of the {TEX[0], C, B} bits from the translation table descriptor. The possible values for each field other than TR6 are:

<table>
<thead>
<tr>
<th>TR&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Device-nGnRnE memory</td>
</tr>
<tr>
<td>0b01</td>
<td>Device-nGnRE memory</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved. The effect of programming a field to 0b11 is CONSTRAINED UNPREDICTABLE, see ‘Unallocated values in fields of AArch32 System registers and translation table entries’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

The meaning of the TR6 field is IMPLEMENTATION DEFINED.
This field resets to an architecturally **UNKNOWN** value.

### Accessing the PRRR

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}\ <\text{coproc}\}, \{#\}<\text{opc1}\>, \ <\text{Rt}\>, \ <\text{CRn}\>, \ <\text{CRm}\}{, \{#\}\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() &amp; !ELUsingAArch32(EL2) &amp; HSTR_EL2.T10 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() &amp; ELUsingAArch32(EL2) &amp; HSTR.T10 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() &amp; !ELUsingAArch32(EL2) &amp; HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() &amp; ELUsingAArch32(EL2) &amp; HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) &amp; ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return PRRR_S;
    else
      return PRRR_NS;
    else
      return PRRR;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) &amp; ELUsingAArch32(EL3) then
      return PRRR_NS;
    else
      return PRRR;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return PRRR_S;
    else
      return PRRR_NS;
\]

\[
\text{MCR}\{<c>\}{<q>}\ <\text{coproc}\}, \{#\}<\text{opc1}\>, \ <\text{Rt}\>, \ <\text{CRn}\>, \ <\text{CRm}\}{, \{#\}\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            PRRR_S = R[t];
        else
            PRRR_NS = R[t];
        end
    else
        PRRR = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PRRR_NS = R[t];
    else
        PRRR = R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            PRRR_S = R[t];
        else
            PRRR_NS = R[t];
        end
    end
REVIDR, Revision ID Register

The REVIDR characteristics are:

**Purpose**

Provides implementation-specific minor revision information.

**Configuration**

AArch32 System register REVIDR bits [31:0] are architecturally mapped to AArch64 System register `REVIDR_EL1[31:0]`.

If REVIDR has the same value as `MIDR`, then its contents have no significance.

**Attributes**

REVIDR is a 32-bit register.

**Field descriptions**

The REVIDR bit assignments are:

```
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

IMPLEMENTATION DEFINED
```

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

**Accessing the REVIDR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

```
coproc | opc1 | CRn | CRm | opc2
-------|------|-----|-----|-----
0b1111  | 0b000| 0b000| 0b000| 0b110
```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return REVIDR;
elsif PSTATE.EL == EL2 then
    return REVIDR;
elsif PSTATE.EL == EL3 then
    return REVIDR;
RMR, Reset Management Register

The RMR characteristics are:

**Purpose**

If EL1 or EL3 is the highest implemented Exception level and this register is implemented:

- A write to the register at the highest implemented Exception level can request a Warm reset.
- If the highest implemented Exception level can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

**Configuration**

AArch32 System register RMR bits [31:0] are architecturally mapped to AArch64 System register RMR_EL1[31:0] when IsHighestEL(EL1).

AArch32 System register RMR bits [31:0] are architecturally mapped to AArch64 System register RMR_EL3[31:0] when HaveEL(EL3).

Only implemented if EL1 or EL3 is the highest implemented Exception level. In this case:

- If the highest implemented Exception level can use AArch32 and AArch64 then this register must be implemented.
- If the highest implemented Exception level cannot use AArch64 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

RMR is a 32-bit register.

**Field descriptions**

The RMR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-20</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
<tr>
<td>19</td>
<td>RR, bit [1]</td>
<td>Reset Request. Setting this bit to 1 requests a Warm reset. This field resets to 0.</td>
</tr>
<tr>
<td>0</td>
<td>AA64, bit [0]</td>
<td>When the highest implemented Exception level can use AArch64, determines which Execution state the PE boots into after a Warm reset:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AA64</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AArch32.</td>
</tr>
<tr>
<td>0b1</td>
<td>AArch64.</td>
</tr>
</tbody>
</table>
On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If the highest implemented Exception level cannot use AArch64 this bit is RAZ/WI.

When implemented as a RW field, this field resets to 0 on a Cold reset.

**Accessing the RMR**

When EL3 is implemented, Arm deprecates accessing this register from any PE mode other than Monitor mode.

Accesses to this register use the following encodings:

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{coproc} & \text{opc1} & \text{CRn} & \text{CRm} & \text{opc2} \\
\hline
0b1111 & 0b000 & 0b1100 & 0b0000 & 0b010 \\
\hline
\end{array}
\]

if \text{PSTATE.EL IN \{EL3, EL1\}} && \text{IsHighestEL(PSTATE.EL)} then

\text{return RMR;}

else

\text{UNDEFINED;}

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{coproc} & \text{opc1} & \text{CRn} & \text{CRm} & \text{opc2} \\
\hline
0b1111 & 0b000 & 0b1100 & 0b0000 & 0b010 \\
\hline
\end{array}
\]

if \text{PSTATE.EL IN \{EL3, EL1\}} && \text{IsHighestEL(PSTATE.EL)} then

\text{RMR = R[t];}

else

\text{UNDEFINED;}
RVBAR, Reset Vector Base Address Register

The RVBAR characteristics are:

**Purpose**

If EL3 is not implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch32 state.

**Configuration**

This register is only implemented if the highest Exception level implemented is capable of using AArch32, and is not EL3.

**Attributes**

RVBAR is a 32-bit register.

**Field descriptions**

The RVBAR bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  |  9  |  8  |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |  RES1

**Bits [31:1]**

Reset Address[31:1]. Bits [31:1] of the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 32-bit state.

**Bit [0]**

Reserved, RES1.

**Accessing the RVBAR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL IN {EL2, EL1} & IsHighestEL(PSTATE.EL) then
    return RVBAR;
else
    UNDEFINED;
```
The SCR characteristics are:

**Purpose**

When EL3 is implemented and can use AArch32, defines the configuration of the current Security state. It specifies:

- The Security state, either Secure or Non-secure.
- What mode the PE branches to if an IRQ, FIQ, or External abort occurs.
- Whether the CPSR.F or CPSR.A bits can be modified when SCR.NS==1.

**Configuration**

AArch32 System register SCR bits [31:0] can be mapped to AArch64 System register SCR_EL3[31:0], but this is not architecturally mandated.

Some or all RW fields of this register have defined reset values. These apply whenever the register is accessible. This means they apply when the PE resets into EL3 using AArch32.

**Attributes**

SCR is a 32-bit register.

**Field descriptions**

The SCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, reserved</td>
</tr>
<tr>
<td>30</td>
<td>TERR</td>
<td>Trap Error record accesses. Generate a Monitor Trap exception on accesses to the following registers from modes other than Monitor mode: ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS. When ARMv8.4-RAS is implemented, ERXMISC4, ERXMISC5, ERXMISC6, ERXMISC7.</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**TERR, bit [15]**

When RAS is implemented:

Trap Error record accesses. Generate a Monitor Trap exception on accesses to the following registers from modes other than Monitor mode:

ERRIDR, ERRSELR, ERXADDR, ERXADDR2, ERXCTLR, ERXCTLR2, ERXFR, ERXFR2, ERXMISC0, ERXMISC1, ERXMISC2, ERXMISC3, and ERXSTATUS. When ARMv8.4-RAS is implemented, ERXMISC4, ERXMISC5, ERXMISC6, ERXMISC7.

<table>
<thead>
<tr>
<th>TERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to the specified registers from modes other than Monitor mode generate a Monitor Trap exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.
Bit [14]

Reserved, RES0.

TWE, bit [13]

Traps WFE instructions to Monitor mode.

<table>
<thead>
<tr>
<th>TWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFE instruction in any mode other than Monitor mode is trapped to Monitor mode, if the instruction would otherwise have caused the PE to enter a low-power state and the attempted execution does not generate an exception that is taken to EL1 or EL2 by SCTLR.nTWE or HCR.TWE. Any exception that is taken to EL1 or to EL2 has priority over this trap.</td>
</tr>
</tbody>
</table>

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

In a system where the PE resets into EL3, this field resets to 0.

TWI, bit [12]

Traps WFI instructions to Monitor mode.

<table>
<thead>
<tr>
<th>TWI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt to execute a WFI instruction in any mode other than Monitor mode is trapped to Monitor mode, if the instruction would otherwise have caused the PE to enter a low-power state and the attempted execution does not generate an exception that is taken to EL1 or EL2 by SCTLR.nTWI or HCR.TWI. Any exception that is taken to EL1 or to EL2 has priority over this trap.</td>
</tr>
</tbody>
</table>

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

In a system where the PE resets into EL3, this field resets to 0.

Bits [11:10]

Reserved, RES0.
**SIF, bit [9]**

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory. The possible values for this bit are:

<table>
<thead>
<tr>
<th>SIF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state instruction fetches from Non-secure memory are permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure state instruction fetches from Non-secure memory are not permitted.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

In a system where the PE resets into EL3, this field resets to 0.

**HCE, bit [8]**

Hypervisor Call instruction enable. If EL2 is implemented, enables execution of HVC instructions at Non-secure EL1 and EL2.

<table>
<thead>
<tr>
<th>HCE</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0 | HVC instructions are:
| | • **UNDEFINED** at Non-secure EL1. The Undefined Instruction exception is taken from PL1 to PL1. |
| | • **UNPREDICTABLE** at EL2. Behavior is one of the following:
| | ◦ The instruction is **UNDEFINED**. |
| | ◦ The instruction executes as a NOP. |
| 0b1 | HVC instructions are enabled at Non-secure EL1 and EL2. |

**Note**

HVC instructions are always **UNDEFINED** at EL0 and in Secure state.

If EL2 is not implemented, this bit is RES0 and HVC is **UNDEFINED**.

In a system where the PE resets into EL3, this field resets to 0.

**SCD, bit [7]**

Secure Monitor Call disable. Disables SMC instructions.

<table>
<thead>
<tr>
<th>SCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SMC instructions are enabled.</td>
</tr>
</tbody>
</table>
| 0b1 | In Non-secure state, SMC instructions are **UNDEFINED**. The Undefined Instruction exception is taken from the current Exception level to the current Exception level. In Secure state, behavior is one of the following:
| | • The instruction is **UNDEFINED**. |
| | • The instruction executes as a NOP. |

**Note**

SMC instructions are always **UNDEFINED** at PL0.

In a system where the PE resets into EL3, this field resets to 0.

**nET, bit [6]**

Not Early Termination. This bit disables early termination. The possible values of this bit are:

<table>
<thead>
<tr>
<th>nET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Early termination permitted. Execution time of data operations can depend on the data values.</td>
</tr>
<tr>
<td>0b1</td>
<td>Disable early termination. The number of cycles required for data operations is forced to be independent of the data values.</td>
</tr>
</tbody>
</table>
This IMPLEMENTATION DEFINED mechanism can disable data dependent timing optimizations from multiplies and data operations. It can provide system support against information leakage that might be exploited by timing correlation types of attack.

On implementations that do not support early termination or do not support disabling early termination, this bit is RES0.

In a system where the PE resets into EL3, this field resets to 0.

**AW, bit [5]**

When the value of SCR.EA is 1 and the value of HCR.AMO is 0, this bit controls whether CPSR.A masks an External abort taken from Non-secure state, and the possible values of this bit are:

<table>
<thead>
<tr>
<th>AW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External aborts taken from Non-secure state are not masked by CPSR.A, and are taken to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>External aborts taken from Secure state are masked by CPSR.A.</td>
</tr>
</tbody>
</table>

When SCR.EA is 0 or HCR.AMO is 1, this bit has no effect.

In a system where the PE resets into EL3, this field resets to 0.

**FW, bit [4]**

When the value of SCR.FIQ is 1 and the value of HCR.FMO is 0, this bit controls whether CPSR.F masks an FIQ interrupt taken from Non-secure state, and the possible values of this bit are:

<table>
<thead>
<tr>
<th>FW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An FIQ taken from Non-secure state is not masked by CPSR.F, and is taken to EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>An FIQ taken from Secure state is masked by CPSR.F.</td>
</tr>
</tbody>
</table>

When SCR.FIQ is 0 or HCR.FMO is 1, this bit has no effect.

In a system where the PE resets into EL3, this field resets to 0.

**EA, bit [3]**

External Abort handler. This bit controls which mode takes External aborts. The possible values of this bit are:

<table>
<thead>
<tr>
<th>EA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External aborts taken to Abort mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>External aborts taken to Monitor mode.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to 0.

**FIQ, bit [2]**

FIQ handler. This bit controls which mode takes FIQ exceptions. The possible values of this bit are:

<table>
<thead>
<tr>
<th>FIQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>FIQs taken to FIQ mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>FIQs taken to Monitor mode.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to 0.

**IRQ, bit [1]**

IRQ handler. This bit controls which mode takes IRQ exceptions. The possible values of this bit are:
### IRQ

<table>
<thead>
<tr>
<th>IRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>IRQs taken to IRQ mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>IRQs taken to Monitor mode.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to 0.

### NS, bit [0]

Non-secure bit. Except when the PE is in Monitor mode, this bit determines the Security state of the PE:

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PE is in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>PE is in Non-secure state.</td>
</tr>
</tbody>
</table>

If the HCR.TGE bit is set, an attempt to change from a Secure PL1 mode to a Non-secure EL1 mode by changing the SCR.NS bit from 0 to 1 results in the SCR.NS bit remaining as 0.

In a system where the PE resets into EL3, this field resets to 0.

### Accessing the SCR

Accesses to this register use the following encodings:

\[
\text{MRC}\{c\}\{q\} \ <\text{coproc}, \ \#<\text{opc1}, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}\}{, \ \#<\text{opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCR;
endif

\[
\text{MCR}\{c\}\{q\} \ <\text{coproc}, \ \#<\text{opc1}, \ <\text{Rt}, \ <\text{CRn}, \ <\text{CRm}\}{, \ \#<\text{opc2}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        UNDEFINED;
    end if;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR = R[t];
SCTLR, System Control Register

The SCTLR characteristics are:

**Purpose**

Provides the top level control of the system, including its memory system.

**Configuration**

AArch32 System register SCTLR bits [31:0] are architecturally mapped to AArch64 System register `SCTLR_EL1[31:0]`.

Some bits in the register are read-only. These bits relate to non-configurable features of an implementation, and are provided for compatibility with previous versions of the architecture.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**

SCTLR is a 32-bit register.

**Field descriptions**

The SCTLR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSSBS</td>
<td>TE</td>
<td>AF</td>
<td>ET</td>
<td>RES0</td>
<td>EE</td>
<td>RES0</td>
<td>SPAN</td>
<td>RES1</td>
<td>RES0</td>
<td>UWXN</td>
<td>WXM</td>
<td>nTWE</td>
<td>RES0</td>
<td>nTWI</td>
<td>RES0</td>
<td>V</td>
<td>RES1</td>
<td>EnRCTX</td>
<td>RES0</td>
<td>SED</td>
<td>ITD</td>
<td>UNK</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DSSBS, bit [31]**

When ARMv8.0-SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

<table>
<thead>
<tr>
<th>DSSBS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PSTATE.SSBS is set to 0 on an exception to any mode in this security state except Hyp mode</td>
</tr>
<tr>
<td>0b1</td>
<td>PSTATE.SSBS is set to 1 on an exception to any mode in this security state except Hyp mode</td>
</tr>
</tbody>
</table>

**Note**

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

This field resets to an IMPLEMENTATION DEFINED value.

**Otherwise:**

Reserved, RES0.
**TE, bit [30]**

T32 Exception Enable. This bit controls whether exceptions to an Exception Level that is executing at PL1 are taken to A32 or T32 state:

<table>
<thead>
<tr>
<th>TE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exceptions, including reset, taken to A32 state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Exceptions, including reset, taken to T32 state.</td>
</tr>
</tbody>
</table>

This field resets to an IMPLEMENTATION DEFINED choice between:

- 0
- A value determined by an input configuration signal.

**AFE, bit [29]**

Access Flag Enable. When using the Short-descriptor translation table format for the PL1&0 translation regime, this bit enables use of the AP[0] bit in the translation descriptors as the Access flag, and restricts access permissions in the translation descriptors to the simplified model. The possible values of this bit are:

<table>
<thead>
<tr>
<th>AFE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>In the translation table descriptors, AP[0] is an access permissions bit. The full range of access permissions is supported. No Access flag is implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>In the translation table descriptors, AP[0] is the Access flag. Only the simplified model for access permissions is supported.</td>
</tr>
</tbody>
</table>

When using the Long-descriptor translation table format, the VMSA behaves as if this bit is set to 1, regardless of the value of this bit.

The AFE bit is permitted to be cached in a TLB.

This field resets to 0.

**TRE, bit [28]**

TEX remap enable. This bit enables remapping of the TEX[2:1] bits in the PL1&0 translation regime for use as two translation table bits that can be managed by the operating system. Enabling this remapping also changes the scheme used to describe the memory region attributes in the VMSA. The possible values of this bit are:

<table>
<thead>
<tr>
<th>TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TEX remap disabled. TEX[2:0] are used, with the C and B bits, to describe the memory region attributes.</td>
</tr>
<tr>
<td>0b1</td>
<td>TEX remap enabled. TEX[2:1] are reassigned for use as bits managed by the operating system. The TEX[0], C, and B bits are used to describe the memory region attributes, with the MMU remap registers.</td>
</tr>
</tbody>
</table>

When the value of TTBCR.EAE is 1, this bit is RES1.

The TRE bit is permitted to be cached in a TLB.

This field resets to 0.

**Bits [27:26]**

Reserved, RES0.

**EE, bit [25]**

The value of the PSTATE.E bit on branch to an exception vector or coming out of reset, and the endianness of stage 1 translation table walks in the PL1&0 translation regime.

The possible values of this bit are:
Meaning

<table>
<thead>
<tr>
<th>EE</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Little-endian. PSTATE.E is cleared to 0 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&amp;0 translation regime are little-endian.</td>
</tr>
<tr>
<td>0b1</td>
<td>Big-endian. PSTATE.E is set to 1 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&amp;0 translation regime are big-endian.</td>
</tr>
</tbody>
</table>

If an implementation does not provide Big-endian support for data accesses at Exception Levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support for data accesses at Exception Levels higher than EL0, this bit is RES1.

This field resets to an IMPLEMENTATION DEFINED choice between:

- 0
- A value determined by an input configuration signal.

**Bit [24]**

Reserved, RES0.

**SPAN, bit [23]**

When ARMv8.1-PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1 from either Secure or Non-secure state, or to EL3 from Secure state when EL3 is using AArch32.

<table>
<thead>
<tr>
<th>SPAN</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0  | CPSR.PAN is set to 1 in the following situations:  

- In Non-secure state, on taking an exception to EL1.  
- In Secure state, when EL3 is using AArch64, on taking an exception to EL1.  
- In Secure state, when EL3 is using AArch32, on taking an exception to EL3. |
| 0b1  | The value of CPSR.PAN is left unchanged on taking an exception to EL1. |

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

**Bit [22]**

Reserved, RES1.

**Bit [21]**

Reserved, RES0.

**UWXN, bit [20]**

Unprivileged write permission implies PL1 XN (Execute-never). This bit can force all memory regions that are writable at PL0 to be treated as XN for accesses from software executing at PL1. The possible values of this bit are:

<table>
<thead>
<tr>
<th>UWXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable at PL0 forced to XN for accesses from software executing at PL1.</td>
</tr>
</tbody>
</table>
The UWXN bit is permitted to be cached in a TLB.

This field resets to 0.

**WXN, bit [19]**

Write permission implies XN (Execute-never). For the PL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN. The possible values of this bit are:

<table>
<thead>
<tr>
<th>WXN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control has no effect on memory access permissions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any region that is writable in the PL1&amp;0 translation regime is forced to XN for accesses from software executing at PL1 or PL0.</td>
</tr>
</tbody>
</table>

This bit applies only when SCTLR.M bit is set.

The WXN bit is permitted to be cached in a TLB.

This field resets to 0.

**nTWE, bit [18]**

Traps EL0 execution of WFE instructions to Undefined mode.

<table>
<thead>
<tr>
<th>nTWE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a WFE instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

This field resets to 1.

**Bit [17]**

Reserved, RES0.

**nTWI, bit [16]**

Traps EL0 execution of WFI instructions to Undefined mode.

<table>
<thead>
<tr>
<th>nTWI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Any attempt to execute a WFI instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>This control does not cause any instructions to be trapped.</td>
</tr>
</tbody>
</table>

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

**Note**

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE of WFI are not guaranteed to be taken, even if the WFE or
WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

This field resets to 1.

**Bits [15:14]**

Reserved, RES0.

**V, bit [13]**

Vectors bit. This bit selects the base address of the exception vectors for exceptions taken to a PE mode other than Monitor mode or Hyp mode:

<table>
<thead>
<tr>
<th>V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal exception vectors. Base address is held in <strong>VBAR</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>High exception vectors (Hivecs), base address 0xFFFF0000. This base address cannot be remapped.</td>
</tr>
</tbody>
</table>

This field resets to an **IMPLEMENTATION DEFINED** choice between:

- 0
- A value determined by an input configuration signal.

**I, bit [12]**

Instruction access Cacheability control, for accesses at EL1 and EL0:

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All instruction access to Normal memory from PL1 and PL0 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&amp;0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>All instruction access to Normal memory from PL1 and PL0 can be cached at all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&amp;0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory.</td>
</tr>
</tbody>
</table>

Instruction accesses to Normal memory from EL1 and EL0 are Cacheable regardless of the value of the SCTLR.I bit if either:

- EL2 is using AArch32 and the value of **HCR.DC** is 1.
- EL2 is using AArch64 and the value of **HCR_EL2.DC** is 1.

This field resets to 0.

**Bit [11]**

Reserved, RES1.

**EnRCTX, bit [10]**

When ARMv8.0-CSV2 is implemented:

Enable EL0 Access to the AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions. The defined values are:

<table>
<thead>
<tr>
<th>EnRCTX</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL0 access to these instructions is disabled, and these instructions are trapped to EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL0 access to these instructions is enabled.</td>
</tr>
</tbody>
</table>
**Note**

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

This field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Bit [9]**

Reserved, RES0.

**SED, bit [8]**

SETEND instruction disable. Disables SETEND instructions at PL0 and PL1.

<table>
<thead>
<tr>
<th>SED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>SETEND instruction execution is enabled at PL0 and PL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>SETEND instructions are <strong>UNDEFINED</strong> at PL0 and PL1.</td>
</tr>
</tbody>
</table>

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

This field resets to 0.

**ITD, bit [7]**

IT Disable. Disables some uses of IT instructions at PL1 and PL0.

<table>
<thead>
<tr>
<th>ITD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All IT instruction functionality is enabled at PL1 and PL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Any attempt at PL1 or PL0 to execute any of the following is <strong>UNDEFINED</strong>:</td>
</tr>
<tr>
<td></td>
<td>• All encodings of the IT instruction with hw1[3:0]! = 1000.</td>
</tr>
<tr>
<td></td>
<td>• All encodings of the subsequent instruction with the following values for hw1:</td>
</tr>
<tr>
<td></td>
<td>◦ 11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM.</td>
</tr>
<tr>
<td></td>
<td>◦ 1011xxxxxxxxxxxx: All instructions in Miscellaneous 16-bit instructions.</td>
</tr>
<tr>
<td></td>
<td>◦ 10100xxxxxxxxxxx: ADD Rd, PC, #imm</td>
</tr>
<tr>
<td></td>
<td>◦ 01001xxxxxxxxxxx: LDR Rd, [PC, #imm]</td>
</tr>
<tr>
<td></td>
<td>◦ 0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC.</td>
</tr>
<tr>
<td></td>
<td>◦ 010001xxx1111xxx: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn.</td>
</tr>
</tbody>
</table>

These instructions are always **UNDEFINED**, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.

It is **IMPLEMENTATION DEFINED** whether the IT instruction is treated as:

- A 16-bit instruction, that can only be followed by another 16-bit instruction.
- The first half of a 32-bit instruction.

This means that, for the situations that are **UNDEFINED**, either the second 16-bit instruction or the 32-bit instruction is **UNDEFINED**. An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.
If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONstrained UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section E1.2.4.

ITD is optional, but if it is implemented in the SCTLRL then it must also be implemented in the SCTLRL_EL1. If it is not implemented then this bit is RAZ/WI.

This field resets to 0.

**UNK, bit [6]**

Writes to this bit are IGNORED. Reads of this bit return an UNKNOWN value.

This field resets to an architecturally UNKNOWN value.

**CP15BEN, bit [5]**

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from PL1 and PL0:

<table>
<thead>
<tr>
<th>CP15BEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISR instructions is UNDEFINED.</td>
</tr>
<tr>
<td>0b1</td>
<td>PL0 and PL1 execution of the CP15DMB, CP15DSB, and CP15ISR instructions is enabled.</td>
</tr>
</tbody>
</table>

CP15BEN is optional, but if it is implemented in the SCTLRL then it must also be implemented in the SCTLRL_EL1. If it is not implemented then this bit is RAZ/WI.

This field resets to 1.

**LSMAOE, bit [4]**

When ARMv8.2-LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

<table>
<thead>
<tr>
<th>LSMAOE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For all memory accesses at EL1 or EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 is as defined for Armv8.0.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

This field resets to 1.

**Otherwise:**

Reserved, RES1.

**nTLSMD, bit [3]**

When ARMv8.2-LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.
### nTLSMD

<table>
<thead>
<tr>
<th><strong>nTLSMD</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped.</td>
</tr>
</tbody>
</table>

This bit is permitted to be cached in a TLB.

This field resets to 1.

**Otherwise:**

Reserved, RES1.

### C, bit [2]

Cacheability control, for data accesses at EL1 and EL0:

<table>
<thead>
<tr>
<th><strong>C</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&amp;0 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&amp;0 stage 1 translation tables, can be cached at all levels of data and unified cache.</td>
</tr>
</tbody>
</table>

The PE ignores SCLTR.C for Non-secure state and data accesses to Normal memory from EL1 and EL0 are Cacheable if either:

- EL2 is using AArch32 and the value of HCR.DC is 1.
- EL2 is using AArch64 and the value of HCR_EL2.DC is 1.

This field resets to 0.

### A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at PL1 and PL0:

<table>
<thead>
<tr>
<th><strong>A</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>0b0</strong></td>
<td>Alignment fault checking disabled when executing at PL1 or PL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed.</td>
</tr>
<tr>
<td><strong>0b1</strong></td>
<td>Alignment fault checking enabled when executing at PL1 or PL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception.</td>
</tr>
</tbody>
</table>

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

This field resets to 0.

### M, bit [0]

MMU enable for EL1 and EL0 stage 1 address translation. Possible values of this bit are:
Meaning

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>EL1 and EL0 stage 1 address translation disabled.</td>
</tr>
<tr>
<td></td>
<td>See the SCTLR.I field for the behavior of instruction accesses to</td>
</tr>
<tr>
<td></td>
<td>Normal memory.</td>
</tr>
<tr>
<td>0b1</td>
<td>EL1 and EL0 stage 1 address translation enabled.</td>
</tr>
</tbody>
</table>

In the Non-secure state the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of the field if either:

- EL2 is using AArch32 and the value of HCR \{DC, TGE\} is not \{0, 0\}.
- EL2 is using AArch64 and the value of HCR_EL2 \{DC, TGE\} is not \{0, 0\}.

This field resets to 0.

**Accessing the SCTLR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then

UNDEFINED;

elsif PSTATE.EL == EL1 then

if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then

AArch64.AArch32SystemAccessTrap(EL2, 0x03);

elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then

AArch32.TakeHypTrapException(0x03);

elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then

AArch64.AArch32SystemAccessTrap(EL2, 0x03);

elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then

AArch32.TakeHypTrapException(0x03);

elsif HaveEL(EL3) && ELUsingAArch32(EL3) then

if SCR.NS == '0' then

return SCTLR_S;

else

return SCTLR_NS;

else

return SCTLR;

elsif PSTATE.EL == EL2 then

if HaveEL(EL3) && ELUsingAArch32(EL3) then

return SCTLR_NS;

else

return SCTLR;

elsif PSTATE.EL == EL3 then

if SCR.NS == '0' then

return SCTLR_S;

else

return SCTLR_NS;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            SCTLR_S = R[t];
        else
            SCTLR_NS = R[t];
        end if;
    else
        SCTLR = R[t];
    end if;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        SCTLR_NS = R[t];
    else
        SCTLR = R[t];
    end if;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            SCTLR_S = R[t];
        else
            SCTLR_NS = R[t];
        end if;
    end if;
SDCR, Secure Debug Control Register

The SDCR characteristics are:

**Purpose**

Provides EL3 configuration options for self-hosted debug, trace, and the Performance Monitors Extension.

**Configuration**

AArch32 System register SDCR bits [31:0] can be mapped to AArch64 System register MDCR_EL3[31:0], but this is not architecturally mandated.

Some or all RW fields of this register have defined reset values. These apply whenever the register is accessible. This means they apply when the PE resets into EL3 using AArch32.

**Attributes**

SDCR is a 32-bit register.

**Field descriptions**

The SDCR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | MTPME | TDCC | RES0 | SCCD | RES0 | EPMAD | EDAD | TTRF | STE | RES0 | SPD | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 |

**Bits [31:29]**

Reserved, RES0.

**MTPME, bit [28]**

When ARMv8.6-MTPMU is implemented:

Multi-threaded PMU Enable. Enables use of the PMEVTYPER<n>.MT bits.

<table>
<thead>
<tr>
<th>MTPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ARMv8.6-MTPMU is disabled. The Effective value of PMEVTYPER&lt;n&gt;.MT is zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>PMEVTYPER&lt;n&gt;.MT bits not affected by this bit.</td>
</tr>
</tbody>
</table>

If ARMv8.6-MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this bit is 0.

On a Cold reset, in a system where the PE resets into EL3, this field resets to 1.

Otherwise:

Reserved, RES0.

**TDCC, bit [27]**

When ARMv8.6-FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel in modes other than Monitor mode to Monitor mode.
TDCC

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This control does not cause any register accesses to be trapped.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to the DCC registers in modes other than Monitor mode generate a Monitor Trap exception, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state.</td>
</tr>
</tbody>
</table>

The DCC registers trapped by this control are:
- `DBGDTRRXext`, `DBGDTRTXext`, `DBGDSRint`, `DBGDCINT`, and, when the PE is in Non-debug state, `DBGDTRRXint` and `DBGDTRTXint`.

When the PE is in Debug state, SDCR.TDCC does not trap any accesses to:
- `DBGDTRRXint` and `DBGDTRTXint`.

In a system where the PE resets into EL3, this field resets to an architecturally `UNKNOWN` value.

**Otherwise:**

Reserved, RES0.

**Bits [26:24]**

Reserved, RES0.

**SCCD, bit [23]**

When ARMv8.5-PMU is implemented:

Secure Cycle Counter Disable. Prohibits `PMCCNTR` from counting in Secure state.

<table>
<thead>
<tr>
<th>SCCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting by <code>PMCCNTR</code> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting by <code>PMCCNTR</code> is prohibited in Secure state.</td>
</tr>
</tbody>
</table>

This bit does not affect the CPU_CYCLES event or any other event that counts cycles.

In a system where the PE resets into EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bit [22]**

Reserved, RES0.

**EPMAD, bit [21]**

When ARMv8.4-Debug is implemented and PMUv3 is implemented:

External Performance Monitors Non-secure access disable. Controls Non-secure access to Performance Monitors registers by an external debugger.

<table>
<thead>
<tr>
<th>EPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure access to the Performance Monitors registers from an external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure access to the Performance Monitors registers from an external debugger is not permitted.</td>
</tr>
</tbody>
</table>

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.
Otherwise, if EL3 is not implemented and the Effective value of SCR.NS is \(0b0\), then the Effective value of this field is \(0b1\).

In a system where the PE resets into EL3, this field resets to \(0\).

**When PMUv3 is implemented:**

External Performance Monitors access disable. Controls access to Performance Monitors registers by an external debugger.

<table>
<thead>
<tr>
<th>EPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0)</td>
<td>Access to Performance Monitors registers from an external debugger is permitted.</td>
</tr>
<tr>
<td>(0b1)</td>
<td>Access to Performance Monitors registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface.</td>
</tr>
</tbody>
</table>

If the Performance Monitors Extension does not support external debug interface accesses this bit is RES0.

Otherwise, if EL3 is not implemented and the Effective value of SCR.NS is \(0b0\), then the Effective value of this field is \(0b1\).

In a system where the PE resets into EL3, this field resets to \(0\).

**Otherwise:**

Reserved, RES0.

**EDAD, bit [20]**

**When ARMv8.4-Debug is implemented:**

External debug Non-secure access disable. Controls Non-secure access to breakpoint, watchpoint, and OSLAR_EL1 registers by an external debugger.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0)</td>
<td>Non-secure access to debug registers from an external debugger is permitted.</td>
</tr>
<tr>
<td>(0b1)</td>
<td>Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 from an external debugger is not permitted.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR.NS is \(0b0\), then the Effective value of this field is \(0b1\).

In a system where the PE resets into EL3, this field resets to \(0\).

**When ARMv8.2-Debug is implemented:**

External debug access disable. Controls access to breakpoint, watchpoint, and OSLAR_EL1 registers by an external debugger.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0)</td>
<td>Access to debug registers from an external debugger is permitted.</td>
</tr>
<tr>
<td>(0b1)</td>
<td>Access to breakpoint registers, watchpoint registers and OSLAR_EL1 from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR.NS is \(0b0\), then the Effective value of this field is \(0b1\).

In a system where the PE resets into EL3, this field resets to \(0\).
Otherwise:

External debug access disable. Controls access to breakpoint, watchpoint, and optionally OSLAR_EL1 registers by an external debugger:

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Access to debug registers from an external debugger is permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access to breakpoint registers and watchpoint registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. It is IMPLEMENTATION DEFINED whether access to the OSLAR_EL1 register from an external debugger is permitted or not permitted.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR.NS is 0b0, then the Effective value of this field is 0b1.

In a system where the PE resets into EL3, this field resets to 0.

**TTRF, bit [19]**

When ARMv8.4-Trace is implemented:

Trap Trace Filter controls. Controls whether accesses at EL2 and EL1 to the trace filter control registers are trapped to EL3.

<table>
<thead>
<tr>
<th>TTRF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to HTRFCR and TRFCR registers are not affected by this control bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>When not in Monitor mode, accesses to HTRFCR and TRFCR registers generate a Monitor Trap exception, unless the access generates a higher priority exception.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

**STE, bit [18]**

When ARMv8.4-Trace is implemented:

Secure Trace Enable. This bit enables tracing in Secure state and controls the level of authentication required by an external debugger to enable external tracing.

<table>
<thead>
<tr>
<th>STE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace is prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace in Secure state is not affected by this bit.</td>
</tr>
</tbody>
</table>

This bit also controls the level of authentication required by an external debugger to enable external tracing. See ‘Register controls to enable self-hosted trace’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

If EL3 is not implemented and the Effective value of SCR.NS is 0b0, the PE behaves as if this bit is set to 0b1.

In a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

**SPME, bit [17]**
**When ARMv8.2-Debug is implemented and PMUv3 is implemented:**

Secure Performance Monitors enable. This allows event counting in Secure state.

<table>
<thead>
<tr>
<th>SPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counting prohibited in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counting allowed in Secure state.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR.NS is 0b0, then the Effective value of this bit is 0b1.

In a system where the PE resets into EL3, this field resets to 0.

**When PMUv3 is implemented:**

Secure Performance Monitors enable. This allows event counting in Secure state.

<table>
<thead>
<tr>
<th>SPME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counting prohibited in Secure state, unless ExternalSecureNoninvasiveDebugEnabled() is TRUE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counting allowed in Secure state.</td>
</tr>
</tbody>
</table>

If EL3 is not implemented and the Effective value of SCR.NS is 0b0, then the Effective value of this bit is 0b1.

In a system where the PE resets into EL3, this field resets to 0.

**Otherwise:**

Reserved, RES0.

**Bit [16]**

Reserved, RES0.

**SPD, bits [15:14]**

AArch32 Secure self-hosted Privileged Debug. Enables or disables debug exceptions from EL3, other than Breakpoint Instruction exceptions

<table>
<thead>
<tr>
<th>SPD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Legacy mode. Debug exceptions from EL3 are enabled by the authentication interface.</td>
</tr>
<tr>
<td>0b10</td>
<td>Secure privileged debug disabled. Debug exceptions from EL3 are disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Secure privileged debug enabled. Debug exceptions from EL3 are enabled.</td>
</tr>
</tbody>
</table>

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is ignored in Non-secure state.

If debug exceptions from EL3 are enabled, then debug exceptions from Secure EL0 are also enabled.

Otherwise, debug exceptions from Secure EL0 are enabled only if the value of SDER.SUIDEN is 0b1.

If EL3 is not implemented and the Effective value of SCR.NS is 0b0, then the Effective value of this field is 0b1.

In a system where the PE resets into EL3, this field resets to 0.
Bits [13:0]
Reserved, RES0.

Accessing the SDCR

Accesses to this register use the following encodings:

\[
\text{MRC\{}<c>\}\{}<q>\} <\text{coproc}, \{\}<\text{opc1}, <\text{Rt}, <\text{CRn}, <\text{CRm}\} \{\}, \{\}<\text{opc2}\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & ELUsingAArch32(EL2) & HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & ELUsingAArch32(EL2) & HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAArch32(EL2) & SCR_EL3.<NS,EEL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL3) & SCR_EL3.NS == '0' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' & CP15SDISABLE2 == HIGH then
    UNDEFINED;
  else
    SDCR = R[t];

\[
\text{MCR\{}<c>\}\{}<q>\} <\text{coproc}, \{\}<\text{opc1}, <\text{Rt}, <\text{CRn}, <\text{CRm}\} \{\}, \{\}<\text{opc2}\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() & ELUsingAArch32(EL2) & HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() & ELUsingAArch32(EL2) & HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif !ELUsingAArch32(EL2) & SCR_EL3.<NS,EEL2> == '01' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif !ELUsingAArch32(EL3) & SCR_EL3.NS == '0' then
    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' & CP15SDISABLE2 == HIGH then
    UNDEFINED;
  else
    SDCR = R[t];
SDER, Secure Debug Enable Register

The SDER characteristics are:

**Purpose**

Controls invasive and non-invasive debug in the Secure EL0 mode.

**Configuration**

AArch32 System register SDER bits [31:0] are architecturally mapped to AArch64 System register SDER32 EL3[31:0].

This register is present only when HaveEL(EL3) or the implemented Security state is Secure state. Otherwise, direct accesses to SDER are UNDEFINED.

This register is ignored by the PE when one or more of the following are true:

- The PE is in Non-secure state.
- EL1 is using AArch64.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SDER is a 32-bit register.

**Field descriptions**

The SDER bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| SUNIDEN | SUIDEN |

**Bits [31:2]**

Reserved, RES0.

**SUNIDEN, bit [1]**

Secure User Non-Invasive Debug Enable.

<table>
<thead>
<tr>
<th>SUNIDEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit does not affect Performance Monitors event counting at Secure EL0</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL3 or EL1 is using AArch32, Performance Monitors event counting is allowed in Secure EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

**SUIDEN, bit [0]**

Secure User Invasive Debug Enable.
**SDER, Secure Debug Enable Register**

<table>
<thead>
<tr>
<th>SUIDEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit does not affect the generation of debug exceptions at Secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>If EL3 or EL1 is using AArch32, debug exceptions from Secure EL0 are enabled.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

## Accessing the SDER

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} \{<opc1>\}, \{<CRn>\}, \{<CRm>\}\{<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b001</td>
<td>0b001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif (!HaveEL(EL3) || !ELUsingAArch32(EL3)) && SCR_EL3.NS == '0' then
    return SDER;
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  return SDER;
end
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b001</td>
<td>0b001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
MCR\{<c>\}{<q>} \{<coproc>\}, \{<opc1>\}, \{<Rt>\}, \{<CRn>\}, \{<CRm>\}\{<opc2>\}
```

```c
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif (!HaveEL(EL3) || !ELUsingAArch32(EL3)) && SCR_EL3.NS == '0' then
    SDER = R[t];
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  UNDEFINED;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
    UNDEFINED;
  else
    SDER = R[t];
```
SPSR, Saved Program Status Register

The SPSR characteristics are:

**Purpose**

Holds the saved process state for the current mode.

**Configuration**

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally **UNKNOWN** values.

**Attributes**

SPSR is a 32-bit register.

**Field descriptions**

The SPSR bit assignments are:

```
  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

**N, bit [31]**

Set to the value of PSTATE.N on taking an exception to the current mode, and copied to PSTATE.N on executing an exception return operation in the current mode.

**Z, bit [30]**

Set to the value of PSTATE.Z on taking an exception to the current mode, and copied to PSTATE.Z on executing an exception return operation in the current mode.

**C, bit [29]**

Set to the value of PSTATE.C on taking an exception to the current mode, and copied to PSTATE.C on executing an exception return operation in the current mode.

**V, bit [28]**

Set to the value of PSTATE.V on taking an exception to the current mode, and copied to PSTATE.V on executing an exception return operation in the current mode.

**Q, bit [27]**

Set to the value of PSTATE.Q on taking an exception to the current mode, and copied to PSTATE.Q on executing an exception return operation in the current mode.

**IT[1:0], bits [26:25]**

IT block state bits for the T32 IT (If-Then) instruction. See IT[7:2] for explanation of this field.
J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass Safe. This bit is set to the value of PSTATE.SSBS on taking an exception to the current mode, and copied to PSTATE.SSBS on executing an exception return operation in the current mode.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. This bit is set to the value of PSTATE.PAN on taking an exception to the current mode, and copied to PSTATE.PAN on executing an exception return operation in the current mode.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When ARMv8.4-DIT is implemented:

Data Independent Timing. This bit is set to the value of PSTATE.DIT on taking an exception to the current mode, and copied to PSTATE.DIT on executing an exception return operation in the current mode.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state bit. Shows the value of PSTATE.IL immediately before the exception was taken.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

IT[7:2], bits [15:10]

IT block state bits for the T32 IT (If-Then) instruction. This field must be interpreted in two parts.

- IT[7:5] holds the base condition for the IT block. The base condition is the top 3 bits of the condition code specified by the first condition field of the IT instruction.
- IT[4:0] encodes the size of the IT block, which is the number of instructions that are to be conditionally executed, by the position of the least significant 1 in this field. It also encodes the value of the least significant bit of the condition code for each instruction in the block.

The IT field is 0b00000000 when no IT block is active.
E, bit [9]

Endianness state bit. Controls the load and store endianness for data accesses:

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Little-endian operation</td>
</tr>
<tr>
<td>0b1</td>
<td>Big-endian operation</td>
</tr>
</tbody>
</table>

Instruction fetches ignore this bit.

If an implementation does not provide Big-endian support, this bit is RES0. If it does not provide Little-endian support, this bit is RES1.

If an implementation provides Big-endian support but only at EL0, this bit is RES0 for an exception return to any Exception level other than EL0.

Likewise, if it provides Little-endian support only at EL0, this bit is RES1 for an exception return to any Exception level other than EL0.

When the reset value of the SCTLR.EE bit is defined by a configuration input signal, that value also applies to the CPSR.E bit on reset, and therefore applies to software execution from reset.

A, bit [8]

SError interrupt mask bit.

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked</td>
</tr>
</tbody>
</table>

I, bit [7]

IRQ mask bit.

<table>
<thead>
<tr>
<th>I</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked</td>
</tr>
</tbody>
</table>

F, bit [6]

FIQ mask bit.

<table>
<thead>
<tr>
<th>F</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception not masked</td>
</tr>
<tr>
<td>0b1</td>
<td>Exception masked</td>
</tr>
</tbody>
</table>

T, bit [5]

T32 Instruction set state bit. Determines the AArch32 instruction set state that the exception was taken from.

<table>
<thead>
<tr>
<th>T</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Taken from A32 state</td>
</tr>
<tr>
<td>0b1</td>
<td>Taken from T32 state</td>
</tr>
</tbody>
</table>

M[4], bit [4]

Execution state that the exception was taken from.

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Exception taken from AArch32</td>
</tr>
</tbody>
</table>
M[3:0], bits [3:0]

AArch32 mode that an exception was taken from.

<table>
<thead>
<tr>
<th>M[3:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>User.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Monitor (only valid in Secure state, if EL3 is implemented and can use AArch32).</td>
</tr>
<tr>
<td>0b0111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b1111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved.

Accessing the SPSR

SPSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions. For more details, see MRS, MSR (register), and MSR (immediate) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
The SPSR_abt characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to Abort mode.

**Configuration**

AArch32 System register SPSR_abt bits [31:0] are architecturally mapped to AArch64 System register SPSR_abt[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_abt is a 32-bit register.

**Field descriptions**

The SPSR_abt bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Abort mode, and copied to PSTATE.Z on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Abort mode, and copied to PSTATE.C on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Abort mode, and copied to PSTATE.V on executing an exception return operation in Abort mode.

This field resets to an architecturally UNKNOWN value.
Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Abort mode, and copied to PSTATE.Q on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Abort mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Abort mode.

On executing an exception return operation in Abort mode SPSR_abt.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**J, bit [24]**

RES0.

In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

**SSBS, bit [23]**

*When ARMv8.0-SSBS is implemented:*

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Abort mode, and copied to PSTATE.SSBS on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**PAN, bit [22]**

*When ARMv8.1-PAN is implemented:*

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Abort mode, and copied to PSTATE.PAN on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**DIT, bit [21]**

*When ARMv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Abort mode, and copied to PSTATE.DIT on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Abort mode, and copied to PSTATE.IL on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Abort mode, and copied to PSTATE.GE on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Abort mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in Abort mode.

**SPSR_abt.IT** must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Abort mode, and copied to PSTATE.E on executing an exception return operation in Abort mode.

If the implementation does not support big-endian operation, SPSR_abt.E is RES0. If the implementation does not support little-endian operation, SPSR_abt.E is RES1. On executing an exception return operation in Abort mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_abt.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_abt.E is RES1.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Abort mode, and copied to PSTATE.A on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Abort mode, and copied to PSTATE.I on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Abort mode, and copied to PSTATE.F on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.
T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Abort mode, and copied to PSTATE.T on executing an exception return operation in Abort mode.

This field resets to an architecturally **UNKNOWN** value.

**M[4:0], bits [4:0]**

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Abort mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Abort mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_abt.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Abort mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_abt**

SPSR_abt is accessible in all modes other than User mode and Abort mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

**MRS{<c>}{<q>} <Rd>, SPSR_abt**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0100</td>
</tr>
</tbody>
</table>

**MSR{<c>}{<q>} SPSR_abt, <Rn>**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0100</td>
</tr>
</tbody>
</table>
SPSR_fiq, Saved Program Status Register (FIQ mode)

The SPSR_fiq characteristics are:

Purpose

Holds the saved process state when an exception is taken to FIQ mode.

Configuration

AArch32 System register SPSR_fiq bits [31:0] are architecturally mapped to AArch64 System register SPSR_fiq[31:0]. RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SPSR_fiq is a 32-bit register.

Field descriptions

The SPSR_fiq bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to FIQ mode, and copied to PSTATE.N on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to FIQ mode, and copied to PSTATE.Z on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to FIQ mode, and copied to PSTATE.C on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to FIQ mode, and copied to PSTATE.V on executing an exception return operation in FIQ mode.

This field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to FIQ mode, and copied to PSTATE.Q on executing an exception return operation in FIQ mode.
SPSR_fiq, Saved Program Status Register (FIQ mode)

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to FIQ mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in FIQ mode.

On executing an exception return operation in FIQ mode SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**J, bit [24]**

RES0.

In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

**SSBS, bit [23]**

When Armv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to FIQ mode, and copied to PSTATE.SSBS on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

When Armv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to FIQ mode, and copied to PSTATE.PAN on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**DIT, bit [21]**

When Armv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to FIQ mode, and copied to PSTATE.DIT on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.
IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to FIQ mode, and copied to PSTATE.IL on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to FIQ mode, and copied to PSTATE.GE on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to FIQ mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in FIQ mode.

SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to FIQ mode, and copied to PSTATE.E on executing an exception return operation in FIQ mode.

If the implementation does not support big-endian operation, SPSR_fiq.E is **RES0**. If the implementation does not support little-endian operation, SPSR_fiq.E is **RES1**. On executing an exception return operation in FIQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_fiq.E is **RES0**, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_fiq.E is **RES1**.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to FIQ mode, and copied to PSTATE.A on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to FIQ mode, and copied to PSTATE.I on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to FIQ mode, and copied to PSTATE.F on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to FIQ mode, and copied to PSTATE.T on executing an exception return operation in FIQ mode.

This field resets to an architecturally **UNKNOWN** value.
**M[4:0], bits [4:0]**

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to FIQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in FIQ mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_fiq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in FIQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_fiq**

SPSR_fiq is accessible in all modes other than User mode and FIQ mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

\[
\text{MRS\{<c>\}<q>\} <Rd>, SPSR_fiq}
\]

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>0b1110</td>
</tr>
</tbody>
</table>

\[
\text{MSR\{<c>\}<q>\} SPSR_fiq, <Rn>}
\]

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>0b1110</td>
</tr>
</tbody>
</table>
SPSR_hyp, Saved Program Status Register (Hyp mode)

The SPSR_hyp characteristics are:

Purpose

Holds the saved process state when an exception is taken to Hyp mode.

Configuration

AArch32 System register SPSR_hyp bits [31:0] are architecturally mapped to AArch64 System register SPSR_EL2[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

Attributes

SPSR_hyp is a 32-bit register.

Field descriptions

The SPSR_hyp bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Hyp mode, and copied to PSTATE.N on executing an exception return operation in Hyp mode.

This field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Hyp mode, and copied to PSTATE.Z on executing an exception return operation in Hyp mode.

This field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Hyp mode, and copied to PSTATE.C on executing an exception return operation in Hyp mode.

This field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Hyp mode, and copied to PSTATE.V on executing an exception return operation in Hyp mode.

This field resets to an architecturally UNKNOWN value.
**Q, bit [27]**

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Hyp mode, and copied to PSTATE.Q on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Hyp mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Hyp mode.

On executing an exception return operation in Hyp mode SPSR_hyp.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**J, bit [24]**

RES0.

In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

**SSBS, bit [23]**

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Hyp mode, and copied to PSTATE.SSBS on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Hyp mode, and copied to PSTATE.PAN on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**DIT, bit [21]**

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Hyp mode, and copied to PSTATE.DIT on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Hyp mode, and copied to PSTATE.IL on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Hyp mode, and copied to PSTATE.GE on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Hyp mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in Hyp mode.

SPSR_hyp.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to Hyp mode, and copied to PSTATE.E on executing an exception return operation in Hyp mode.

If the implementation does not support big-endian operation, SPSR_hyp.E is RES0. If the implementation does not support little-endian operation, SPSR_hyp.E is RES1. On executing an exception return operation in Hyp mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_hyp.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_hyp.E is RES1.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Hyp mode, and copied to PSTATE.A on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Hyp mode, and copied to PSTATE.I on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.

**F, bit [6]**

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Hyp mode, and copied to PSTATE.F on executing an exception return operation in Hyp mode.

This field resets to an architecturally **UNKNOWN** value.
T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Hyp mode, and copied to PSTATE.T on executing an exception return operation in Hyp mode.

This field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Hyp mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Hyp mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11010</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b11111</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_hyp.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Hyp mode is an illegal return event, as described in ‘Illegal return events from AArch32 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Accessing the SPSR_hyp

SPSR_hyp is accessible only in Monitor mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

**MRS{<c>}{<q>} <Rd>, SPSR_hyp**

```
<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b1110</td>
</tr>
</tbody>
</table>
```

**MSR{<c>}{<q>} SPSR_hyp, <Rn>**

```
<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b1110</td>
</tr>
</tbody>
</table>
```
SPSR_irq, Saved Program Status Register (IRQ mode)

The SPSR_irq characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to IRQ mode.

**Configuration**

AArch32 System register SPSR_irq bits [31:0] are architecturally mapped to AArch64 System register SPSR_irq[31:0]. RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_irq is a 32-bit register.

**Field descriptions**

The SPSR_irq bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to IRQ mode, and copied to PSTATE.N on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to IRQ mode, and copied to PSTATE.Z on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to IRQ mode, and copied to PSTATE.C on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to IRQ mode, and copied to PSTATE.V on executing an exception return operation in IRQ mode.

This field resets to an architecturally UNKNOWN value.

**Q, bit [27]**

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to IRQ mode, and copied to PSTATE.Q on executing an exception return operation in IRQ mode.
This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to IRQ mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in IRQ mode.

On executing an exception return operation in IRQ mode SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**J, bit [24]**

RES0.

In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

**SSBS, bit [23]**

When Armv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to IRQ mode, and copied to PSTATE.SSBS on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

When Armv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to IRQ mode, and copied to PSTATE.PAN on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**DIT, bit [21]**

When Armv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to IRQ mode, and copied to PSTATE.DIT on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.
IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to IRQ mode, and copied to PSTATE.IL on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to IRQ mode, and copied to PSTATE.GE on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to IRQ mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in IRQ mode.

SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to IRQ mode, and copied to PSTATE.E on executing an exception return operation in IRQ mode.

If the implementation does not support big-endian operation, SPSR_irq.E is **RES0**. If the implementation does not support little-endian operation, SPSR_irq.E is **RES1**. On executing an exception return operation in IRQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_irq.E is **RES0**, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_irq.E is **RES1**.

This field resets to an architecturally **UNKNOWN** value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to IRQ mode, and copied to PSTATE.A on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to IRQ mode, and copied to PSTATE.I on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to IRQ mode, and copied to PSTATE.F on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to IRQ mode, and copied to PSTATE.T on executing an exception return operation in IRQ mode.

This field resets to an architecturally **UNKNOWN** value.
M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to IRQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in IRQ mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11111</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_irq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in IRQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Accessing the SPSR_irq

SPSR_irq is accessible in all modes other than User mode and IRQ mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

**MRS{<c>}{<q>} <Rd>, SPSR_irq**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0000</td>
</tr>
</tbody>
</table>

**MSR{<c>}{<q>} SPSR_irq, <Rn>**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
SPSR_mon, Saved Program Status Register (Monitor mode)

The SPSR_mon characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to Monitor mode.

**Configuration**

AArch32 System register SPSR_mon bits [31:0] can be mapped to AArch64 System register $SPSR_{EL3}[31:0]$, but this is not architecturally mandated.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_mon is a 32-bit register.

**Field descriptions**

The SPSR_mon bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Monitor mode, and copied to PSTATE.N on executing an exception return operation in Monitor mode.

This field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Monitor mode, and copied to PSTATE.Z on executing an exception return operation in Monitor mode.

This field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Monitor mode, and copied to PSTATE.C on executing an exception return operation in Monitor mode.

This field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Monitor mode, and copied to PSTATE.V on executing an exception return operation in Monitor mode.

This field resets to an architecturally UNKNOWN value.
Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Monitor mode, and copied to PSTATE.Q on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Monitor mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Monitor mode.

On executing an exception return operation in Monitor mode SPSR_mon.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**J, bit [24]**

RES0.

In previous versions of the architecture, the \{J, T\} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

**SSBS, bit [23]**

*When Armv8.0-SSBS is implemented:*

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Monitor mode, and copied to PSTATE.SSBS on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**PAN, bit [22]**

*When Armv8.1-PAN is implemented:*

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Monitor mode, and copied to PSTATE.PAN on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

*Otherwise:*

Reserved, RES0.

**DIT, bit [21]**

*When Armv8.4-DIT is implemented:*

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Monitor mode, and copied to PSTATE.DIT on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

**IL, bit [20]**

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Monitor mode, and copied to PSTATE.IL on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

**GE, bits [19:16]**

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Monitor mode, and copied to PSTATE.GE on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[7:2], bits [15:10]**

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Monitor mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in Monitor mode.

SPSR_mon.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**E, bit [9]**

Endianness. Set to the value of PSTATE.E on taking an exception to Monitor mode, and copied to PSTATE.E on executing an exception return operation in Monitor mode.

If the implementation does not support big-endian operation, SPSR_mon.E is RES0. If the implementation does not support little-endian operation, SPSR_mon.E is RES1. On executing an exception return operation in Monitor mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_mon.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_mon.E is RES1.

This field resets to an architecturally **UNKNOWN** value.

**A, bit [8]**

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Monitor mode, and copied to PSTATE.A on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

**I, bit [7]**

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Monitor mode, and copied to PSTATE.I on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

**F, bit [6]**

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Monitor mode, and copied to PSTATE.F on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.
T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Monitor mode, and copied to PSTATE.T on executing an exception return operation in Monitor mode.

This field resets to an architecturally **UNKNOWN** value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Monitor mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Monitor mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>User.</td>
</tr>
<tr>
<td>0b0001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Monitor.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Hyp.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b1111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_mon.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Monitor mode is an illegal return event, as described in ‘Illegal return events from AArch32 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_mon**

SPSR_mon is only accessible in EL3 modes other than Monitor mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

MRS{<c>}{<q>} <Rd>, SPSR_mon

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b1100</td>
</tr>
</tbody>
</table>

MSR{<c>}{<q>} SPSR_mon, <Rn>

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b1100</td>
</tr>
</tbody>
</table>
SPSR_svc, Saved Program Status Register (Supervisor mode)

The SPSR_svc characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to Supervisor mode.

**Configuration**

AArch32 System register SPSR_svc bits [31:0] are architecturally mapped to AArch64 System register SPSR_EL1[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_svc is a 32-bit register.

**Field descriptions**

The SPSR_svc bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**N, bit [31]**

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Supervisor mode, and copied to PSTATE.N on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

**Z, bit [30]**

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Supervisor mode, and copied to PSTATE.Z on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

**C, bit [29]**

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Supervisor mode, and copied to PSTATE.C on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

**V, bit [28]**

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Supervisor mode, and copied to PSTATE.V on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.
Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Supervisor mode, and copied to PSTATE.Q on executing an exception return operation in Supervisor mode.

This field resets to an architecturally **UNKNOWN** value.

**IT[1:0], bits [26:25]**

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Supervisor mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Supervisor mode.

On executing an exception return operation in Supervisor mode SPSR_svc.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally **UNKNOWN** value.

**J, bit [24]**

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

**SSBS, bit [23]**

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Supervisor mode, and copied to PSTATE.SSBS on executing an exception return operation in Supervisor mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**PAN, bit [22]**

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Supervisor mode, and copied to PSTATE.PAN on executing an exception return operation in Supervisor mode.

This field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**DIT, bit [21]**

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Supervisor mode, and copied to PSTATE.DIT on executing an exception return operation in Supervisor mode.

This field resets to an architecturally **UNKNOWN** value.
Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Supervisor mode, and copied to PSTATE.IL on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Supervisor mode, and copied to PSTATE.GE on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Supervisor mode, and copied to PSTATE.IT[7:2] on executing an exception return operation in Supervisor mode.

SPSR_svc.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Supervisor mode, and copied to PSTATE.E on executing an exception return operation in Supervisor mode.

If the implementation does not support big-endian operation, SPSR_svc.E is RES0. If the implementation does not support little-endian operation, SPSR_svc.E is RES1. On executing an exception return operation in Supervisor mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_svc.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_svc.E is RES1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Supervisor mode, and copied to PSTATE.A on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Supervisor mode, and copied to PSTATE.I on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Supervisor mode, and copied to PSTATE.F on executing an exception return operation in Supervisor mode.

This field resets to an architecturally UNKNOWN value.
**T, bit [5]**

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Supervisor mode, and copied to PSTATE.T on executing an exception return operation in Supervisor mode.

This field resets to an architecturally **UNKNOWN** value.

**M[4:0], bits [4:0]**

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Supervisor mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Supervisor mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_svc.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Supervisor mode is an illegal return event, as described in 'Illegal return events from AArch32 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the SPSR_svc**

SPSR_svc is accessible in all modes other than User mode and Supervisor mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

**MRS{<c>}{<q>} <Rd>, SPSR_svc**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0010</td>
</tr>
</tbody>
</table>

**MSR{<c>}{<q>} SPSR_svc, <Rn>**

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0010</td>
</tr>
</tbody>
</table>
The SPSR_und characteristics are:

**Purpose**

Holds the saved process state when an exception is taken to Undefined mode.

**Configuration**

AArch32 System register SPSR_und bits [31:0] are architecturally mapped to AArch64 System register SPSR_und[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

SPSR_und is a 32-bit register.

**Field descriptions**

The SPSR_und bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>N</td>
<td>Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Undefined mode, and copied to PSTATE.N on executing an exception return operation in Undefined mode. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>30</td>
<td>Z</td>
<td>Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Undefined mode, and copied to PSTATE.Z on executing an exception return operation in Undefined mode. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>29</td>
<td>C</td>
<td>Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Undefined mode, and copied to PSTATE.C on executing an exception return operation in Undefined mode. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>28</td>
<td>V</td>
<td>Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Undefined mode, and copied to PSTATE.V on executing an exception return operation in Undefined mode. This field resets to an architecturally UNKNOWN value.</td>
</tr>
</tbody>
</table>
Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Undefined mode, and copied to PSTATE.Q on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

IT[1:0], bits [26:25]

If-Then. Set to the value of PSTATE.IT[1:0] on taking an exception to Undefined mode, and copied to PSTATE.IT[1:0] on executing an exception return operation in Undefined mode.

On executing an exception return operation in Undefined mode SPSR_und.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state. Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When ARMv8.0-SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Undefined mode, and copied to PSTATE.SSBS on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When ARMv8.1-PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Undefined mode, and copied to PSTATE.PAN on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When ARMv8.4-DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Undefined mode, and copied to PSTATE.DIT on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Undefined mode, and copied to
PSTATE.IL on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Undefined mode, and copied to
PSTATE.GE on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

IT[7:2], bits [15:10]

If-Then. Set to the value of PSTATE.IT[7:2] on taking an exception to Undefined mode, and copied to PSTATE.IT[7:2]
on executing an exception return operation in Undefined mode.

SPSR_und.IT must contain a value that is valid for the instruction being returned to.

This field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Undefined mode, and copied to PSTATE.E on
executing an exception return operation in Undefined mode.

If the implementation does not support big-endian operation, SPSR_und.E is RES0. If the implementation does not
support little-endian operation, SPSR_und.E is RES1. On executing an exception return operation in Undefined mode, if
the implementation does not support big-endian operation at the Exception level being returned to, SPSR_und.E is
RES0, and if the implementation does not support little-endian operation at the Exception level being returned to,
SPSR_und.E is RES1.

This field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Undefined mode, and copied to PSTATE.A on
executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Undefined mode, and copied to PSTATE.I
on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Undefined mode, and copied to PSTATE.F
on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.
T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Undefined mode, and copied to PSTATE.T on executing an exception return operation in Undefined mode.

This field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Undefined mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Undefined mode.

<table>
<thead>
<tr>
<th>M[4:0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10000</td>
<td>User.</td>
</tr>
<tr>
<td>0b10001</td>
<td>FIQ.</td>
</tr>
<tr>
<td>0b10010</td>
<td>IRQ.</td>
</tr>
<tr>
<td>0b10011</td>
<td>Supervisor.</td>
</tr>
<tr>
<td>0b10111</td>
<td>Abort.</td>
</tr>
<tr>
<td>0b11011</td>
<td>Undefined.</td>
</tr>
<tr>
<td>0b11111</td>
<td>System.</td>
</tr>
</tbody>
</table>

Other values are reserved. If SPSR_und.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Undefined mode is an illegal return event, as described in ‘Illegal return events from AArch32 state’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

Accessing the SPSR_und

SPSR_und is accessible in all modes other than User mode and Undefined mode. For more details, see MRS (banked register) and MSR (banked register) in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Accesses to this register use the following encodings:

MRS{<c>}{<q>} <Rd>, SPSR_und

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0110</td>
</tr>
</tbody>
</table>

MSR{<c>}{<q>} SPSR_und, <Rn>

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0110</td>
</tr>
</tbody>
</table>
TCMTR, TCM Type Register

The TCMTR characteristics are:

**Purpose**

Provides information about the implementation of the TCM.

**Configuration**

If EL1 or above can use AArch32 then this register must be implemented.

**Attributes**

TCMTR is a 32-bit register.

**Field descriptions**

The TCMTR bit assignments are:

<table>
<thead>
<tr>
<th>Bit 31</th>
<th>Bit 30</th>
<th>Bit 29</th>
<th>Bit 28</th>
<th>Bit 27</th>
<th>Bit 26</th>
<th>Bit 25</th>
<th>Bit 24</th>
<th>Bit 23</th>
<th>Bit 22</th>
<th>Bit 21</th>
<th>Bit 20</th>
<th>Bit 19</th>
<th>Bit 18</th>
<th>Bit 17</th>
<th>Bit 16</th>
<th>Bit 15</th>
<th>Bit 14</th>
<th>Bit 13</th>
<th>Bit 12</th>
<th>Bit 11</th>
<th>Bit 10</th>
<th>Bit 9</th>
<th>Bit 8</th>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED.

**Accessing the TCMTR**

Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
   UNDEFINED;
elif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T0 == '1' then
      AArch32.TakeHypTrapException(0x03);
elif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TID1 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elif EL2Enabled() && ELUsingAAArch32(EL2) && HCR.TID1 == '1' then
      AArch32.TakeHypTrapException(0x03);
   else
      return TCMTR;
elif PSTATE.EL == EL2 then
   return TCMTR;
elif PSTATE.EL == EL3 then
   return TCMTR;
TLBIALL, TLB Invalidate All

The TLBIALL characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.

- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.

- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIALL is a 32-bit System instruction.

**Field descriptions**

TLBIALL ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBIALL instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAAArch32(EL2) && HCR.FB == '1' then
    TLBIALLIS();
  else
    TLBIALL();
elsif PSTATE.EL == EL2 then
  TLBIALL();
elsif PSTATE.EL == EL3 then
  TLBIALL();
The TLBIALLH characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIALLH is a 32-bit System instruction.

**Field descriptions**

TLBIALLH ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBIALLH instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is **CONSTRAINED UNPREDICTABLE**, and one of the following behaviors must occur:

- The instruction is **UNDEFINED**.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

```
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
</tr>
</tbody>
</table>

```python
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        TLBIALLH();
    elsif PSTATE.EL == EL3 then
        if !HaveEL(EL2) then
            UNDEFINED;
        else
            TLBIALLH();
    ```
TLBIALLHIS, TLB Invalidate All, Hyp mode, Inner Shareable

The TLBIALLHIS characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIALLHIS is a 32-bit System instruction.

**Field descriptions**

TLBIALLHIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBIALLHIS instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBIALLHIS();
elsif PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    UNDEFINED;
  else
    TLBIALLHIS();
else
The TLBIALLIS characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
  - Would be required for the EL1&0 translation regime.
  - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2 and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIALLIS is a 32-bit System instruction.

**Field descriptions**

TLBIALLIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBIALLIS instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c}\}{<q}\ <\text{coproc}\}, \{#\}<\text{opc1}\}, <\text{Rt}> , <\text{CRn}>, <\text{CRm}\} , \{#\}<\text{opc2}\} \\
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    TLBIALLIS();
elsif PSTATE.EL == EL2 then
  TLBIALLIS();
elsif PSTATE.EL == EL3 then
  TLBIALLIS();
TLBIALLNSNH, TLB Invalidate All, Non-Secure Non-Hyp

The TLBIALLNSNH characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIALLNSNH is a 32-bit System instruction.

**Field descriptions**

TLBIALLNSNH ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBIALLNSNH instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>} <\text{coproc}>, \{#<\text{opc1}\}, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\}{, \{#<\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBIALLNSNH();
elsif PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    UNDEFINED;
  else
    TLBIALLNSNH();
else
  UNDEFINED;
TLBIALLNSNHIS, TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

The TLBIALLNSNHIS characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIALLNSNHIS is a 32-bit System instruction.

**Field descriptions**

TLBIALLNSNHIS ignores the value in the register specified by the instruction encoding. Software does not have to write a value to the register before issuing this instruction.

**Executing the TLBIALLNSNHIS instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONstrained UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b100</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsf EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
else
  UNDEFINED;
elsf PSTATE.EL == EL2 then
  TLBIALLNSNHIS();
elsf PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    UNDEFINED;
else
    TLBIALLNSNHIS();
```
TLBIASID, TLB Invalidate by ASID match

The TLBIASID characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIASID is a 32-bit System instruction.

**Field descriptions**

The TLBIASID input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>ASID</td>
</tr>
</tbody>
</table>

Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

**Executing the TLBIASID instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIASIDIS(R[t]);
    else
        TLBIASID(R[t]);
    end if
elsif PSTATE.EL == EL2 then
    TLBIASID(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIASID(R[t]);
TLBIASIDIS, TLB Invalidate by ASID match, Inner Shareable

The TLBIASIDIS characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
  - Is from a level of lookup above the final level.
  - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIASIDIS is a 32-bit System instruction.

**Field descriptions**

The TLBIASIDIS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>ASID</td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**ASID, bits [7:0]**

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

**Executing the TLBIASIDIS instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>\} <\text{coproc}, \{#}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}\}, \{#}\langle\text{opc2}\rangle
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        TLBIASIDIS(R[t]);
    end
elsif PSTATE.EL == EL2 then
    TLBIASIDIS(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIASIDIS(R[t]);
The TLBIIPAS2 characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

*Note*

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIIPAS2 is a 32-bit System instruction.

**Field descriptions**

The TLBIIPAS2 input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | IPA[39:12] |

**Bits [31:28]**

Reserved, RES0.

**IPA[39:12], bits [27:0]**

Bits[39:12] of the intermediate physical address to match.

**Executing the TLBIIPAS2 instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.
Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}\{<q>\} <\text{coproc}, \{\#<opc1>, <Rt>, <CRn>, <CRm}\{, \{\#<opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
else
  if PSTATE.EL == EL2 then
    TLBIIPAS2(R[t]);
  elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
      UNDEFINED;
    elsif SCR.NS == '0' then
      //no operation
    else
      TLBIIPAS2(R[t]);
  else
    UNDEFINED;
  end if;
else
  if PSTATE.EL == EL2 then
    TLBIIPAS2(R[t]);
  elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
      UNDEFINED;
    elsif SCR.NS == '0' then
      //no operation
    else
      TLBIIPAS2(R[t]);
  else
    UNDEFINED;
  end if;
end if;
The TLBIIPAS2IS characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- `SCR.NS` is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Note**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIIPAS2IS is a 32-bit System instruction.

**Field descriptions**

The TLBIIPAS2IS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>...</td>
</tr>
<tr>
<td>30</td>
<td>...</td>
</tr>
<tr>
<td>29</td>
<td>...</td>
</tr>
<tr>
<td>28</td>
<td>...</td>
</tr>
<tr>
<td>27</td>
<td>...</td>
</tr>
<tr>
<td>26</td>
<td>...</td>
</tr>
<tr>
<td>25</td>
<td>...</td>
</tr>
<tr>
<td>24</td>
<td>...</td>
</tr>
<tr>
<td>23</td>
<td>...</td>
</tr>
<tr>
<td>22</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>...</td>
</tr>
<tr>
<td>19</td>
<td>...</td>
</tr>
<tr>
<td>18</td>
<td>...</td>
</tr>
<tr>
<td>17</td>
<td>...</td>
</tr>
<tr>
<td>16</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>...</td>
</tr>
<tr>
<td>13</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>RES0</td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**IPA[39:12]**

Bits[39:12] of the intermediate physical address to match.

**Executing the TLBIIPAS2IS instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.
Accesses to this instruction use the following encodings:

\[
\text{MCR}\{\text{<c>}}\{\text{<q>}}\ \{\text{coproc}\}, \{\#\text{opc1}\}, \text{<Rt>}, \text{<CRn>}, \text{<CRm>}, \{\#\text{opc2}\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>

```c
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    endif
else
    if PSTATE.EL == EL2 then
        TLBIIPAS2(R[t]);
    elseif PSTATE.EL == EL3 then
        if !HaveEL(EL2) then
            UNDEFINED;
        elseif SCR.NS == '0' then
            //no operation
        else
            TLBIIPAS2(R[t]);
        endif
    endif
endif
```

27/09/2019 18:22; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The TLBIIPAS2L characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- SCR.NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Note**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIIPAS2L is a 32-bit System instruction.

**Field descriptions**

The TLBIIPAS2L input value bit assignments are:

<p>| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES0</td>
<td>IPA[39:12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**IPA[39:12], bits [27:0]**

Bits[39:12] of the intermediate physical address to match.

**Executing the TLBIIPAS2L instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.
Accesses to this instruction use the following encodings:

\[
\text{MCR}\{\text{c}1}\{\text{q}\}\{\text{coproc}\}, \{\text{#}\}\text{opc1}, \{\text{Rt}\}, \{\text{CRn}\}, \{\text{CRm}\}\{\{\text{#}\}\text{opc2}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b101</td>
</tr>
</tbody>
</table>

```
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  else
    UNDEFINED;
  endif
else
  UNDEFINED;
endif
```

```
elseif PSTATE.EL == EL2 then
  TLBIIPAS2(R[t]);
elseif PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    UNDEFINED;
  elseif SCR.NS == '0' then
    //no operation
  else
    TLBIIPAS2(R[t]);
  endif
endif
```
The TLBIIPAS2LIS characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- SCR NS is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Note**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIIPAS2LIS is a 32-bit System instruction.

**Field descriptions**

The TLBIIPAS2LIS input value bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>IPA[39:12]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**IPA[39:12], bits [27:0]**

Bits[39:12] of the intermediate physical address to match.

**Executing the TLBIIPAS2LIS instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.
Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c}\}{<q}\} \{<\text{coproc}\}, \{#\}<\text{opc1}\}, <\text{Rt}\}, <\text{CRn}\}, <\text{CRm}\}{, \{#\}<\text{opc2}\}\
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
  end
elsif PSTATE.EL == EL2 then
  TLBIIPAS2IS(R[t]);
elsif PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    UNDEFINED;
  elsif SCR.NS == '0' then
    //no operation
  else
    TLBIIPAS2IS(R[t]);
  end
TLBIMVA, TLB Invalidate by VA

The TLBIMVA characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIMVA is a 32-bit System instruction.

**Field descriptions**

The TLBIMVA input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>ASID</td>
</tr>
</tbody>
</table>

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this operation.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

**Executing the TLBIMVA instruction**

Accesses to this instruction use the following encodings:
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
    TLBIMVAIS(R[t]);
  else
    TLBIMVA(R[t]);
elsif PSTATE.EL == EL2 then
  TLBIMVA(R[t]);
elsif PSTATE.EL == EL3 then
  TLBIMVA(R[t]);
TLBIMVAA, TLB Invalidate by VA, All ASID

The TLBIMVAA characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIMVAA is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAA input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-12</td>
<td>VA</td>
<td>Bits [31:12] Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.</td>
</tr>
<tr>
<td>11-0</td>
<td>Reserved, RES0</td>
<td>Bits [11:0]</td>
</tr>
</tbody>
</table>

**Executing the TLBIMVAA instruction**

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
    TLBIMVAAIS(R[t]);
else
  TLBIMVAA(R[t]);
elsif PSTATE.EL == EL2 then
  TLBIMVAA(R[t]);
elsif PSTATE.EL == EL3 then
  TLBIMVAA(R[t]);
The TLBIMVAAIS characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIMVAAIS is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAAIS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA</td>
</tr>
<tr>
<td>30</td>
<td>Bits [31:12]</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.</td>
</tr>
<tr>
<td>27</td>
<td>Bits [11:0]</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Executing the TLBIMVAAIS instruction**

Accesses to this instruction use the following encodings:

\[
\text{MCR} \{\text{<c>}\} \{\text{<q>}\} \ <\text{coproc}> , \{\text{<opc1>}, \#\text{<opc2>}}, \ <\text{Rt}> , \ <\text{CRn}>, \ <\text{CRm}>, \{\text{<opc2>}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    TLBIMVAIAS(R[t]);
elsif PSTATE.EL == EL2 then
  TLBIMVAIAS(R[t]);
eselif PSTATE.EL == EL3 then
  TLBIMVAIAS(R[t]);
TLBIMVAAL, TLB Invalidate by VA, All ASID, Last level

The TLBIMVAAL characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Note**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIMVAAL is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAAL input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VA | RES0 |

**VA, bits [31:12]**

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

**Bits [11:0]**

Reserved, RES0.

**Executing the TLBIMVAAL instruction**

Accesses to this instruction use the following encodings:
MCR{<c>}{<q>}{<coproc>}{<opc1>},{<Rt>},{<CRn>},{<CRm}>{,#}{<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
    TLBIMVAALIS(R[t]);
  else
    TLBIMVAAL(R[t]);
elsif PSTATE.EL == EL2 then
  TLBIMVAAL(R[t]);
else PSTATE.EL == EL3 then
  TLBIMVAAL(R[t]);
TLBIMVAALIS, TLB Invalidate by VA, All ASID, Last level, Inner Shareable

The TLBIMVAALIS characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Note**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIMVAALIS is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAALIS input value bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   | VA | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**VA, bits [31:12]**

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

**Bits [11:0]**

Reserved, RES0.

**Executing the TLBIMVAALIS instruction**

Accesses to this instruction use the following encodings:
MCR{<c>}{<q>}{<coproc>}, {#}{<opc1>}, <Rt>, <CRn>, <CRm>{, {#}{<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b111</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    TLBIMVAALIS(R[t]);
elsif PSTATE.EL == EL2 then
  TLBIMVAALIS(R[t]);
elsif PSTATE.EL == EL3 then
  TLBIMVAALIS(R[t]);
TLBIMVAH, TLB Invalidate by VA, Hyp mode

The TLBIMVAH characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIMVAH is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAH input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VA | RES0 |

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:0]**

Reserved, RES0.

**Executing the TLBIMVAH instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c}\}{<q}> \{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm}\{, \{#<opc2>\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
      AArch32.TakeHypTrapException(0x03);
   else
      UNDEFINED;
elsif PSTATE.EL == EL2 then
   TLBIMVAH(R[t]);
elsif PSTATE.EL == EL3 then
   if !HaveEL(EL2) then
      UNDEFINED;
   else
      TLBIMVAH(R[t]);
TLBIMVAHIS, TLB Invalidate by VA, Hyp mode, Inner Shareable

The TLBIMVAHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

Attributes

TLBIMVAHIS is a 32-bit System instruction.

Field descriptions

The TLBIMVAHIS input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VA | RES0 |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBIMVAHIS(R[t]);
elsif PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    UNDEFINED;
  else
    TLBIMVAHIS(R[t]);

The TLBIMVAIS characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is from a level of lookup above the final level and matches the specified ASID.
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

**Attributes**

TLBIMVAIS is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAIS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>ASID</td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:8]**

Reserved, RES0.

**ASID, bits [7:0]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
Executing the TLBIMVAIS instruction

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}\{<q>\} <\text{coproc}>, \{#<opc1>, <Rt>, <\text{CRn}>, <\text{CRm}\}, \{#<opc2>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
      AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
      AArch32.TakeHypTrapException(0x03);
    else
      TLBIMVAIS(R[t]);
    endif
elsif PSTATE.EL == EL2 then
    TLBIMVAIS(R[t]);
elsif PSTATE.EL == EL3 then
    TLBIMVAIS(R[t]);
 TLBIMVAL, TLB Invalidate by VA, Last level

The TLBIMVAL characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIMVAL is a 32-bit System instruction.

**Field descriptions**

The TLBIMVAL input value bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>VA</td>
<td>RES0</td>
<td>ASID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:8]**

Reserved, RES0.

**ASID, bits [7:0]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
Executing the TLBIMVAL instruction

Accesses to this instruction use the following encodings:

MCR{<c>{<q>}, {#}<opc1>, <Rt>, <CRn}, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.FB == '1' then
        TLBIMVALIS(R[t]);
    else
        TLBIMVAL(R[t]);
    elsif PSTATE.EL == EL2 then
        TLBIMVAL(R[t]);
    elsif PSTATE.EL == EL3 then
        TLBIMVAL(R[t]);
The TLBIMVALH characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level
of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate
the specified address.

The invalidation only applies to the PE that executes this System instruction.

**Configuration**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIMVALH is a 32-bit System instruction.

**Field descriptions**

The TLBIMVALH input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VA | RES0 |

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System
instruction.

**Bits [11:0]**

Reserved, RES0.

**Executing the TLBIMVALH instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED
UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

\[
\text{MCR} \{<c>\} \{<q>\} \langle\text{coproc}\rangle, \{#\}<\text{opc1}\rangle, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}\rangle
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && ![ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1'] then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  TLBIMVALH(R[t]);
elsif PSTATE.EL == EL3 then
  if ![HaveEL(EL2) then
    UNDEFINED;
  else
    TLBIMVALH(R[t]);
The TLBIMVALHIS characteristics are:

**Purpose**

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIMVALHIS is a 32-bit System instruction.

**Field descriptions**

The TLBIMVALHIS input value bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>VA</td>
</tr>
<tr>
<td>30</td>
<td>VA</td>
</tr>
<tr>
<td>29</td>
<td>VA</td>
</tr>
<tr>
<td>28</td>
<td>VA</td>
</tr>
<tr>
<td>27</td>
<td>VA</td>
</tr>
<tr>
<td>26</td>
<td>VA</td>
</tr>
<tr>
<td>25</td>
<td>VA</td>
</tr>
<tr>
<td>24</td>
<td>VA</td>
</tr>
<tr>
<td>23</td>
<td>VA</td>
</tr>
<tr>
<td>22</td>
<td>VA</td>
</tr>
<tr>
<td>21</td>
<td>VA</td>
</tr>
<tr>
<td>20</td>
<td>VA</td>
</tr>
<tr>
<td>19</td>
<td>VA</td>
</tr>
<tr>
<td>18</td>
<td>VA</td>
</tr>
<tr>
<td>17</td>
<td>VA</td>
</tr>
<tr>
<td>16</td>
<td>VA</td>
</tr>
<tr>
<td>15</td>
<td>VA</td>
</tr>
<tr>
<td>14</td>
<td>VA</td>
</tr>
<tr>
<td>13</td>
<td>VA</td>
</tr>
<tr>
<td>12</td>
<td>VA</td>
</tr>
<tr>
<td>11</td>
<td>VA</td>
</tr>
<tr>
<td>10</td>
<td>VA</td>
</tr>
<tr>
<td>9</td>
<td>VA</td>
</tr>
<tr>
<td>8</td>
<td>VA</td>
</tr>
<tr>
<td>7</td>
<td>RES0</td>
</tr>
<tr>
<td>6</td>
<td>RES0</td>
</tr>
<tr>
<td>5</td>
<td>RES0</td>
</tr>
<tr>
<td>4</td>
<td>RES0</td>
</tr>
<tr>
<td>3</td>
<td>RES0</td>
</tr>
<tr>
<td>2</td>
<td>RES0</td>
</tr>
<tr>
<td>1</td>
<td>RES0</td>
</tr>
<tr>
<td>0</td>
<td>RES0</td>
</tr>
</tbody>
</table>

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:0]**

Reserved, RES0.

**Executing the TLBIMVALHIS instruction**

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    endif
elsif PSTATE.EL == EL2 then
    TLBIMVALHIS(R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        TLBIMVALHIS(R[t]);
    endif
TLBIMVALID, TLB Invalidate by VA, Last level, Inner Shareable

The TLBIMVALID characteristics are:

**Purpose**

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
  - The entry is a global entry from the final level of lookup.
  - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

**Configuration**

This System instruction is not implemented in architecture versions before Armv8.

**Attributes**

TLBIMVALID is a 32-bit System instruction.

**Field descriptions**

The TLBIMVALID input value bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| VA | RES0 | ASID |

**VA, bits [31:12]**

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

**Bits [11:8]**

Reserved, RES0.

**ASID, bits [7:0]**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
Executing the TLBIMVALIS instruction

Accesses to this instruction use the following encodings:

\[
\text{MCR}\{<c>\}{<q>}\ <\text{coproc}>, \{#\}<\text{opc1}>, \ <\text{Rt}>, \ <\text{CRn}>, \ <\text{CRm}>{{}\{, \{#\}<\text{opc2}\}}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TTLB == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    TLBIMVAL(R[t]);
elsif PSTATE.EL == EL2 then
  TLBIMVAL(R[t]);
elsif PSTATE.EL == EL3 then
  TLBIMVAL(R[t]);
The TLBTR characteristics are:

**Purpose**

Provides information about the TLB implementation. The register must define whether the implementation provides separate instruction and data TLBs, or a unified TLB. Normally, the IMPLEMENTATION DEFINED information in this register includes the number of lockable entries in the TLB.

**Configuration**

**Attributes**

TLBTR is a 32-bit register.

**Field descriptions**

The TLBTR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED, bits [31:1]</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>nU, bit [0]</td>
</tr>
</tbody>
</table>

nU, bit [0]

Not Unified TLB. Indicates whether the implementation has a unified TLB:

<table>
<thead>
<tr>
<th>nU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unified TLB.</td>
</tr>
<tr>
<td>0b1</td>
<td>Separate Instruction and Data TLBs.</td>
</tr>
</tbody>
</table>

**Accessing the TLBTR**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>} <\text{coproc}>, \{#\}<\text{opc1}>, <\text{Rt}>, <\text{CRn}>, <\text{CRm}>\{, \{#\}<\text{opc2}>\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return TLBTR;
    endif
elsif PSTATE.EL == EL2 then
    return TLBTR;
elsif PSTATE.EL == EL3 then
    return TLBTR;
else
    return TLBTR;
eendif
TPIDRPRW, PL1 Software Thread ID Register

The TPIDRPRW characteristics are:

**Purpose**

Provides a location where software executing at EL1 or higher can store thread identifying information that is not visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch32 System register TPIDRPRW bits [31:0] are architecturally mapped to AArch64 System register TPIDR_EL1[31:0].

**Note**

The PE never updates this register.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TPIDRPRW is a 32-bit register.

**Field descriptions**

The TPIDRPRW bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Thread ID. Thread identifying information stored by software running at this Exception level. This field resets to an architecturally UNKNOWN value.</td>
</tr>
</tbody>
</table>

**Accessing the TPIDRPRW**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elself EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
elself HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TPIDRPRW_S;
elself
            return TPIDRPRW_NS;
else
    return TPIDRPRW;
elsif PSTATE.EL == EL2 then
if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return TPIDRPRW_NS;
elself
    return TPIDRPRW;
elsif PSTATE.EL == EL3 then
if SCR.NS == '0' then
    return TPIDRPRW_S;
elself
    return TPIDRPRW_NS;
else
    return TPIDRPRW_NS;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b100</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elself EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
elself HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TPIDRPRW_S = R[t];
elself
            TPIDRPRW_NS = R[t];
else
    TPIDRPRW = R[t];
elself PSTATE.EL == EL2 then
if HaveEL(EL3) && ELUsingAArch32(EL3) then
    TPIDRPRW_NS = R[t];
elself
    TPIDRPRW = R[t];
elself PSTATE.EL == EL3 then
if SCR.NS == '0' then
    TPIDRPRW_S = R[t];
elself
    TPIDRPRW_NS = R[t];
TPIDRURO, PL0 Read-Only Software Thread ID Register

The TPIDRURO characteristics are:

**Purpose**

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch32 System register TPIDRURO bits [31:0] are architecturally mapped to AArch64 System register TPIDRRO_EL0[31:0].

**Note**

The PE never updates this register.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TPIDRURO is a 32-bit register.

**Field descriptions**

The TPIDRURO bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Thread ID</td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Thread ID. Thread identifying information stored by software running at this Exception level.

This field resets to an architecturally UNKNOWN value.

**Accessing the TPIDRURO**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1'
    then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1'
      then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
      SCR_EL3.FGTen == '1') && HFRGR_EL2.TPIDRRO_EL0 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
      return TPIDRURO;
  else
    return TPIDRURO;
  elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1'
      then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1'
      then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
      if SCR.NS == '0' then
        return TPIDRURO_S;
      else
        return TPIDRURO_NS;
    else
      return TPIDRURO_NS;
  else
    return TPIDRURO_NS;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
      return TPIDRURO_NS;
    else
      return TPIDRURO_NS;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return TPIDRURO_S;
    else
      return TPIDRURO_NS;
else
  return TPIDRURO_NS;
end if

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1'
    then
      AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1'
      then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
      if SCR.NS == '0' then
        TPIDRURO_S = R[t];
      else
        TPIDRURO_NS = R[t];
    else
      TPIDRURO = R[t];
  else
    TPIDRURO = R[t];
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    TPIDRURO_NS = R[t];
  else
    TPIDRURO = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    TPIDRURO_S = R[t];
  else
    TPIDRURO_NS = R[t];
else
  TPIDRURO_NS = R[t];

TPIDRURO, PL0 Read-Only Software Thread ID Register
TPIDRURW, PL0 Read/Write Software Thread ID Register

The TPIDRURW characteristics are:

**Purpose**

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

**Configuration**

AArch32 System register TPIDRURW bits [31:0] are architecturally mapped to AArch64 System register TPIDR_EL0[31:0].

**Note**

The PE never updates this register.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TPIDRURW is a 32-bit register.

**Field descriptions**

The TPIDRURW bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Thread ID</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Thread ID. Thread identifying information stored by software running at this Exception level.

This field resets to an architecturally UNKNOWN value.

**Accessing the TPIDRURW**

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}\ <\text{coproc}>\,\{\#<\text{opc1}>\},\ <\text{Rt}>\,\ <\text{CRn}>\,\ <\text{CRm}>\{\,\{\#<\text{opc2}>\}\}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFRTR_EL2.TPIDR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return TPIDRURW;
    end
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TPIDRURW_S;
        else
            return TPIDRURW_NS;
        end
    else
        return TPIDRURW;
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRURW_NS;
    else
        return TPIDRURW;
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TPIDRURW_S;
    else
        return TPIDRURW_NS;
    end
else
    return TPIDRURW;
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>

TPIDRURW, PL0 Read/Write Software Thread ID Register
if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
        SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        TPIDRURW = R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            if SCR.NS == '0' then
                TPIDRURW_S = R[t];
            else
                TPIDRURW_NS = R[t];
            end;
    else
        TPIDRURW = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            TPIDRURW_NS = R[t];
        else
            TPIDRURW = R[t];
        end;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            TPIDRURW_S = R[t];
        else
            TPIDRURW_NS = R[t];
        end;
    end;
end;
TRFCR, Trace Filter Control Register

The TRFCR characteristics are:

**Purpose**

Provides EL1 controls for Trace.

**Configuration**

AArch32 System register TRFCR bits [31:0] are architecturally mapped to AArch64 System register TRFCR_EL1[31:0].

This register is present only when Armv8.4-Trace is implemented. Otherwise, direct accesses to TRFCR are UNDEFINED.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TRFCR is a 32-bit register.

**Field descriptions**

The TRFCR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td>TS</td>
</tr>
</tbody>
</table>

**Bits [31:7]**

Reserved, RES0.

**TS, bits [6:5]**

Timestamp Control. Controls which timebase is used for trace timestamps.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>Virtual timestamp. The traced timestamp is the physical counter value, minus the value of CNTVOFF.</td>
<td>When ARMv8.6-ECV is implemented</td>
</tr>
<tr>
<td>0b10</td>
<td>Guest Physical timestamp. The traced timestamp is the physical counter value, minus the value of CNTPOFF_EL2.</td>
<td>When ARMv8.6-ECV is implemented</td>
</tr>
<tr>
<td>0b11</td>
<td>Physical timestamp. The traced timestamp is the physical counter value.</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- HTRFCR.TS is not 0b00.

When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if either of the following are true:
• **CNTHCTL_EL2** ECV is 0.
• **SCR_EL3** ECVEn is 0.
• **HCR_EL2** {E2H, TGE} is {1, 1}.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [4:2]**

Reserved, **RES0**.

**E1TRE, bit [1]**

EL1 Trace Enable.

<table>
<thead>
<tr>
<th>E1TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing is prohibited in PL1 modes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing is allowed in PL1 modes.</td>
</tr>
</tbody>
</table>

This field is ignored if SelfHostedTraceEnabled() == FALSE.

On a Warm reset, this field resets to 0.

**E0TRE, bit [0]**

EL0 Trace Enable.

<table>
<thead>
<tr>
<th>E0TRE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing is prohibited at EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing is allowed at EL0.</td>
</tr>
</tbody>
</table>

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and enabled in the current security state and **HCR.TGE** == 1.

On a Warm reset, this field resets to 0.

**Accessing the TRFCR**

Accesses to this register use the following encodings:

\[ \text{MRC}\{<c>\}{<q>} \text{<coproc>, } \{#\}\text{<opc1>, }<\text{Rt}>, <\text{CRn}>, <\text{CRm}>, \{#\}<\text{opc2}>\]
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCR.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TTRF == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
        AArch32.TakeMonitorTrapException();
    else
        return TRFCR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return TRFCR;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            UNDEFINED;
        else
            return TRFCR;
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCR.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TTRF == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        AArch32.TakeMonitorTrapException();
else
    TRFCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        AArch32.TakeMonitorTrapException();
else
    TRFCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
else
    TRFCR = R[t];
TTBCR, Translation Table Base Control Register

The TTBCR characteristics are:

**Purpose**

The control register for stage 1 of the PL1&0 translation regime. Its controls include:

- Where the VA range is split between addresses translated using TTBR0 and addresses translated using TTBR1.
- The translation table format used by this stage of translation.

From Armv8.2, when the value of TTBCR.{EAE, T2E} is {1, 1}, TTBCR is used with TTBCR2.

**Configuration**

AArch32 System register TTBCR bits [31:0] are architecturally mapped to AArch64 System register TCR_EL1[31:0].

The current translation table format determines which format of the register is used.

Some RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 then:

- The EAE bit resets to 0 in both the Secure and the Non-secure instances of the register.
- Other reset values apply only to the Secure instance of the register.

**Attributes**

TTBCR is a 32-bit register.

**Field descriptions**

The TTBCR bit assignments are:

**When TTBCR.EAE == 0:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| EAE | RES0 | PD1 | PD0 | RES0 | N |

**EAE, bit [31]**

Extended Address Enable. The meanings of the possible values of this bit are:

<table>
<thead>
<tr>
<th>EAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>Use the VMSAv8-32 translation system with the Short-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Bits [30:6]**

Reserved, RES0.

**PD1, bit [5]**

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1. The encoding of this bit is:
**PD1**

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0  Perform translation table walks using TTBR1.</td>
</tr>
<tr>
<td>0b1  A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**PD0, bit [4]**

Translation table walk disable for translations using TTBR0. This bit controls whether a translation table walk is performed on a TLB miss for an address that is translated using TTBR0. The encoding of this bit is:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0  Perform translation table walks using TTBR0.</td>
</tr>
<tr>
<td>0b1  A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Bit [3]**

Reserved, RES0.

**N, bits [2:0]**

Indicate the width of the base address held in TTBR0. In TTBR0, the base address field is bits[31:14-N]. The value of N also determines:

- Whether TTBR0 or TTBR1 is used as the base address for translation table walks.
- The size of the translation table pointed to by TTBR0.

N can take any value from 0 to 7, that is, from 0b000 to 0b111.

When N has its reset value of 0, the translation table base is compatible with Armv5 and Armv6.

This field resets to 0.

**When TTBCR.EAE == 1:**

**EAE, bit [31]**

Extended Address Enable. The meanings of the possible values of this bit are:

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1  Use the VMSAv8-32 translation system with the Long-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**IMPLEMENTATION DEFINED, bit [30]**

IMPLEMENTATION DEFINED.

This field resets to 0.
SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using TTBR1. Defined values are:

<table>
<thead>
<tr>
<th>SH1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to 0.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using TTBR1.

<table>
<thead>
<tr>
<th>ORGN1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
</tbody>
</table>
| 0b01   | Normal memory, Outer Write-Back Read-Allocate Write-
          Allocate Cacheable.                                  |
| 0b10   | Normal memory, Outer Write-Through Read-Allocate No Write-
          Allocate Cacheable.                                  |
| 0b11   | Normal memory, Outer Write-Back Read-Allocate No Write-
          Allocate Cacheable.                                  |

This field resets to 0.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using TTBR1.

<table>
<thead>
<tr>
<th>IRGN1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
</tbody>
</table>
| 0b01   | Normal memory, Inner Write-Back Read-Allocate Write-
          Allocate Cacheable.                                  |
| 0b10   | Normal memory, Inner Write-Through Read-Allocate No Write-
          Allocate Cacheable.                                  |
| 0b11   | Normal memory, Inner Write-Back Read-Allocate No Write-
          Allocate Cacheable.                                  |

This field resets to 0.

EPD1, bit [23]

Translation table walk disable for translations using TTBR1. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using TTBR1. The encoding of this bit is:

<table>
<thead>
<tr>
<th>EPD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Perform translation table walks using TTBR1.</td>
</tr>
</tbody>
</table>
| 0b1    | A TLB miss on an address that is translated using TTBR1
          generates a Translation fault. No translation table walk
          is performed.                                     |

This field resets to 0.

A1, bit [22]

Selects whether TTBR0 or TTBR1 defines the ASID. The encoding of this bit is:

<table>
<thead>
<tr>
<th>A1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TTBR0 ASID defines the ASID.</td>
</tr>
<tr>
<td>0b1</td>
<td>TTBR1 ASID defines the ASID.</td>
</tr>
</tbody>
</table>
This field resets to 0.

**Bits [21:19]**

Reserved, RES0.

**T1SZ, bits [18:16]**

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long Descriptor translation table format' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for how TTBCR, {T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using TTBR0 and TTBR1.

This field resets to 0.

**Bits [15:14]**

Reserved, RES0.

**SH0, bits [13:12]**

Shareability attribute for memory associated with translation table walks using TTBR0.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to 0.

**ORGN0, bits [11:10]**

Outer cacheability attribute for memory associated with translation table walks using TTBR0.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**IRGN0, bits [9:8]**

Inner cacheability attribute for memory associated with translation table walks using TTBR0.

<table>
<thead>
<tr>
<th>IRGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to 0.
**EPD0, bit [7]**

Translation table walk disable for translations using **TTBR0**. This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using **TTBR0**. The encoding of this bit is:

<table>
<thead>
<tr>
<th>EPD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Perform translation table walks using <strong>TTBR0</strong>.</td>
</tr>
<tr>
<td>0b1</td>
<td>A TLB miss on an address that is translated using <strong>TTBR0</strong> generates a Translation fault. No translation table walk is performed.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**T2E, bit [6]**

When **ARMv8.2-AA32HPD** is implemented:

TTBCR2 Enable.

<table>
<thead>
<tr>
<th>T2E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>TTBCR2</strong> is disabled. The contents of <strong>TTBCR2</strong> are treated as 0 for all purposes other than reading or writing the register.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>TTBCR2</strong> is enabled.</td>
</tr>
</tbody>
</table>

If **TTBCR.EAE==0**, then the behavior is as if the bit is 0.

Otherwise:

Reserved, **RES0**.

**Bits [5:3]**

Reserved, **RES0**.

**T0SZ, bits [2:0]**

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for how **TTBCR**.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using **TTBR0** and **TTBR1**.

This field resets to 0.

**Accessing the TTBCR**

Accesses to this register use the following encodings:

\[
\text{MRC\{<c>\}{<q>} <coproc>, {},<opc1>,<Rt>,<CRn>,<CRm>{, {},<opc2>}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAAArch32(EL3) then
        if SCR.NS == '0' then
            return TTBCR_S;
        else
            return TTBCR_NS;
    else
        return TTBCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAAArch32(EL3) then
        return TTBCR_NS;
    else
        return TTBCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBCR_S;
    else
        return TTBCR_NS;

MCR{<c>}{<q}> {coproc}, {#<opc1>, <Rt>, <CRn>, <CRm>{, {#<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      TTBCR_S = R[t];
    else
      TTBCR_NS = R[t];
    else
      TTBCR = R[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
      TTBCR_NS = R[t];
    else
      TTBCR = R[t];
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
      UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
      UNDEFINED;
    else
      if SCR.NS == '0' then
        TTBCR_S = R[t];
      else
        TTBCR_NS = R[t];

TTBCR2, Translation Table Base Control Register 2

The TTBCR2 characteristics are:

**Purpose**

The second control register for stage 1 of the PL1&0 translation regime.

If ARMv8.2-AA32HPD is not implemented then this register is not implemented and its encoding is UNDEFINED. Otherwise:

- When the value of TTBCR, {EAE, T2E} is not {1, 1} the contents of TTBCR2 are treated as zero for all purposes other than reading or writing the register.
- When the value of TTBCR, {EAE, T2E} is {1, 1} TTBCR2 is used with TTBCR.

**Configuration**

AArch32 System register TTBCR2 bits [31:0] are architecturally mapped to AArch64 System register $TCR_EL1[63:32]$.

This register is present only when ARMv8.2-AA32HPD is implemented. Otherwise, direct accesses to TTBCR2 are UNDEFINED.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TTBCR2 is a 32-bit register.

**Field descriptions**

The TTBCR2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>HWU162</td>
<td>HWU161</td>
<td>HWU160</td>
<td>HWU159</td>
<td>HWU062</td>
<td>HWU061</td>
<td>HWU060</td>
<td>HWU059</td>
<td>HPD1</td>
<td>HPD0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td>RES0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:19]**

Reserved, RES0.

**HWU162, bit [18]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR1.

<table>
<thead>
<tr>
<th>HWU162</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR1, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR1, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**HWU161, bit [17]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR1.

<table>
<thead>
<tr>
<th>HWU161</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR1, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR1, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU160, bit [16]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR1.

<table>
<thead>
<tr>
<th>HWU160</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR1, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR1, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU159, bit [15]**

*When ARMv8.2-TTPBHA is implemented:*

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR1.
### HWU159

<table>
<thead>
<tr>
<th>Meaning</th>
<th>0b0</th>
<th>For translations using TTBR1, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0b1</td>
<td>For translations using TTBR1, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES0.

### HWU062, bit [14]

**When ARMv8.2-TTPBHA is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using TTBR0.

<table>
<thead>
<tr>
<th>HWU062</th>
<th>Meaning</th>
<th>0b0</th>
<th>For translations using TTBR0, bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0b1</td>
<td>For translations using TTBR0, bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES0.

### HWU061, bit [13]

**When ARMv8.2-TTPBHA is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using TTBR0.

<table>
<thead>
<tr>
<th>HWU061</th>
<th>Meaning</th>
<th>0b0</th>
<th>For translations using TTBR0, bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0b1</td>
<td>For translations using TTBR0, bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES0.
HWU060, bit [12]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using TTBR0.

<table>
<thead>
<tr>
<th>HWU060</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR0, bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR0, bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0. This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU059, bit [11]

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using TTBR0.

<table>
<thead>
<tr>
<th>HWU059</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>For translations using TTBR0, bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>For translations using TTBR0, bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1.</td>
</tr>
</tbody>
</table>

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of TTBCR.T2E is 0. This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD1, bit [10]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by TTBR1.

<table>
<thead>
<tr>
<th>HPD1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled if TTBCR.T2E == 1.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are 0.

The Effective value of this field is 0 if the value of TTBCR.T2E is 0. This field resets to an architecturally UNKNOWN value.
HPD0, bit [9]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by TTBR0.

<table>
<thead>
<tr>
<th>HPD0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Hierarchical permissions are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Hierarchical permissions are disabled if TTBCR.T2E == 1.</td>
</tr>
</tbody>
</table>

When disabled, the permissions are treated as if the bits are 0.

The Effective value of this field is 0 if the value of TTBCR.T2E is 0.

This field resets to an architecturally UNKNOWN value.

Bits [8:0]

Reserved, RES0.

Accessing the TTBCR2

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && !HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return TTBCR2_S;
    else
      return TTBCR2_NS;
    end
  else
    return TTBCR2;
  end
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return TTBCR2_NS;
  else
    return TTBCR2;
  end
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return TTBCR2_S;
  else
    return TTBCR2_NS;
else
  return TTBCR2_NS;
end

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b011</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBCR2_S = R[t];
        else
            TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' & CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' & CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR2_S = R[t];
        else
            TTBCR2_NS = R[t];
    end

TTBR0, Translation Table Base Register 0

The TTBR0 characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the PL1&0 translation regime, and other information for this translation regime.

**Configuration**

AArch32 System register TTBR0 bits [63:0] are architecturally mapped to AArch64 System register TTBR0_EL1[63:0].

TTBCR.EAE determines which TTBR0 format is used:

- TTBCR.EAE == 0b0: 32-bit format is used. TTBR0[63:32] are ignored.
- TTBCR.EAE == 0b1: 64-bit format is used.

When EL3 is using AArch32, write access to TTBR0(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

Used in conjunction with the TTBCR. When the 64-bit TTBR0 format is used, cacheability and shareability information is held in the TTBCR, not in TTBR0.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TTBR0 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

**Field descriptions**

The TTBR0 bit assignments are:

**When TTBCR.EAE == 0:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>TTB0</td>
<td>RGN[0]</td>
<td>NOS</td>
<td>RGN</td>
<td>IMP</td>
<td>S</td>
<td>RGN[1]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:32]

Reserved, RES0.

**TTB0, bits [31:7]**

Translation table base address, bits[31:x], where x is 14-(TTBCR.N). Register bits [x-1:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

This field resets to an architecturally UNKNOWN value.
**IRGN[0], bit [6]**

This field is bit[0] of IRGN[1:0].

Inner region bits. Bits [0,6] of this register together indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

<table>
<thead>
<tr>
<th>IRGN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>01</td>
<td>Normal memory, Inner Write-Back Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>10</td>
<td>Normal memory, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>11</td>
<td>Normal memory, Inner Write-Back no Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

**Note**

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for ARMv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[0] is TTBR0[6].
- IRGN[1] is TTBR0[0].

This field resets to an architecturally UNKNOWN value.

**NOS, bit [5]**

Not Outer Shareable. When the value of TTBR0.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

<table>
<thead>
<tr>
<th>NOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory is Outer Shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is Inner Shareable.</td>
</tr>
</tbody>
</table>

This bit is ignored when the value of TTBR0.S is 0.

This field resets to an architecturally UNKNOWN value.

**RGN, bits [4:3]**

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

<table>
<thead>
<tr>
<th>RGN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Normal memory, Outer Write-Back Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back no Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**IMP, bit [2]**

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

This field resets to an architecturally UNKNOWN value.

**S, bit [1]**

Shareable. Indicates whether the memory associated with the translation table walks is Non-shareable:
Meaning

<table>
<thead>
<tr>
<th>S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory is Non-shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is shareable. The TTBR0.NOS field indicates whether the memory is Inner Shareable or Outer Shareable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally `UNKNOWN` value.

**IRGN[1], bit [0]**

This field is bit[1] of IRGN[1:0].

See IRGN[0] for the field description.

**When TTBCR.EAE == 1:**

<table>
<thead>
<tr>
<th>Bits [63:56]</th>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BADDR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CnP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [63:56]

Reserved, RES0.

**ASID, bits [55:48]**

An ASID for the translation table base address. The TTBCR.A1 field selects either TTBR0.ASID or TTBR1.ASID.

This field resets to an architecturally `UNKNOWN` value.

**BADDR, bits [47:1]**

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are constrained unpredictable, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of TTBCR.T0SZ as follows:

- If TTBCR.T0SZ is 0 or 1, x = 5 - TTBCR.T0SZ.
- If TTBCR.T0SZ is greater than 1, x = 14 - TTBCR.T0SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

This field resets to an architecturally `UNKNOWN` value.

**CnP, bit [0]**

When ARMv8.2-TTCNP is implemented:

Common not Private. When TTBCR.EAE == 1, this bit indicates whether each entry that is pointed to by TTBR0 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0.CnP is 1.
The translation table entries pointed to by this instance of TTBR0, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR0 for other PEs in the Inner Shareable domain. This is not affected by:

- The value of TTBR0.CnP on those other PEs.
- The value of TTBCR.EAE on those other PEs.
- The value of the current ASID or, for the Non-secure instance of TTBR0, the value of the current VMID.

The translation table entries pointed to by this instance of TTBR0 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0.CnP is 1 for this instance of TTBR0 and all of the following apply:

- The translation table entries are pointed to by this instance of TTBR0.
- The value of the applicable TTBCR.EAE field is 1.
- The ASID is the same as the current ASID.
- For the Non-secure instance of TTBR0, the VMID is the same as the current VMID.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

**Note**

If the value of the TTBR0.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Accessing the TTBR0**

Accesses to this register use the following encodings:

\[ \text{MRC}\{\langle c\rangle}\{\langle q\rangle\} \text{<coproc>, \{#<opc1>, <Rt>, <CRn>, <CRm>{, \{#<opc2>\}}) \]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return TTBR0_S<31:0>;
    else
      return TTBR0_NS<31:0>;
    else
      return TTBR0<31:0>;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
      return TTBR0_NS<31:0>;
    else
      return TTBR0<31:0>;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return TTBR0_S<31:0>;
    else
      return TTBR0_NS<31:0>;

MCR{<c>{<q}> <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm}{, {#}<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBR0_S = ZeroExtend(R[t]);
        else
            TTBR0_NS = ZeroExtend(R[t]);
    else
        TTBR0 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = ZeroExtend(R[t]);
    else
        TTBR0 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR0_S = ZeroExtend(R[t]);
        else
            TTBR0_NS = ZeroExtend(R[t]);

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
    AArch32.TakeHypTrapException(0x04);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return TTBR0_S;
    else
      return TTBR0_NS;
  else
    return TTBR0;
elsif PSTATE.EL == EL2 then
  if HaveEL(EL3) && ELUsingAArch32(EL3) then
    return TTBR0_NS;
  else
    return TTBR0;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    return TTBR0_S;
  else
    return TTBR0_NS;

MCRR{<coproc>}<CRm>, {#<opc1>}, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBR0_S = R[t2]:R[t];
        else
            TTBR0_NS = R[t2]:R[t];
        end
    else
        TTBR0 = R[t2]:R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = R[t2]:R[t];
    else
        TTBR0 = R[t2]:R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR0_S = R[t2]:R[t];
        else
            TTBR0_NS = R[t2]:R[t];
        end
    end
TTBR1, Translation Table Base Register 1

The TTBR1 characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the PL1&0 translation regime, and other information for this translation regime.

**Configuration**

AArch32 System register TTBR1 bits [63:0] are architecturally mapped to AArch64 System register TTBR1_EL1[63:0]. TTBCR.EAE determines which TTBR1 format is used:

- TTBCR.EAE == 0b0: 32-bit format is used. TTBR1[63:32] are ignored.
- TTBCR.EAE == 0b1: 64-bit format is used.

Used in conjunction with the TTBCR. When the 64-bit TTBR1 format is used, cacheability and shareability information is held in the TTBCR, not in TTBR1.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

TTBR1 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

**Field descriptions**

The TTBR1 bit assignments are:

**When TTBCR.EAE == 0:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | TTB1 | RG[1] | NOS | RG1 | MP | S | RG[0] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

**TTB1, bits [31:7]**

Translation table base address, bits[31:14]. Register bits [13:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [13:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

This field resets to an architecturally UNKNOWN value.
IRGN[1], bit [6]

This field is bit[1] of IRGN[1:0].

Inner region bits. IRGN[1:0] indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

<table>
<thead>
<tr>
<th>IRGN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back no Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

Note

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for Armv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[1] is TTBR1[6].
- IRGN[0] is TTBR1[0].

This field resets to an architecturally UNKNOWN value.

NOS, bit [5]

Not Outer Shareable. When the value of TTBR1.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

<table>
<thead>
<tr>
<th>NOS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory is Outer Shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is Inner Shareable.</td>
</tr>
</tbody>
</table>

This bit is ignored when the value of TTBR1.S is 0.

This field resets to an architecturally UNKNOWN value.

RGN, bits [4:3]

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

<table>
<thead>
<tr>
<th>RGN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back no Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

IMP, bit [2]

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

This field resets to an architecturally UNKNOWN value.

S, bit [1]

Shareable. Indicates whether the memory associated with the translation table walks is Non-shareable:
Meaning

<table>
<thead>
<tr>
<th>S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Memory is Non-shareable.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is shareable. The TTBR1.NOS field indicates whether the</td>
</tr>
<tr>
<td></td>
<td>memory is Inner Shareable or Outer Shareable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**IRGN[0], bit [0]**

This field is bit[0] of IRGN[1:0].

See IRGN[1] for the field description.

**When TTBCR.EAE == 1:**

The field layout is as follows:

<table>
<thead>
<tr>
<th>Bits [63:56]</th>
<th>Bits [47:1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>ASID</td>
</tr>
<tr>
<td>BADDR</td>
<td></td>
</tr>
</tbody>
</table>

Reserved, RES0.

**ASID, bits [55:48]**

An ASID for the translation table base address. The TTBCR.A1 field selects either TTBR0.ASID or TTBR1.ASID.

This field resets to an architecturally **UNKNOWN** value.

**BADDR, bits [47:1]**

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are **CONSTRAINED UNPREDICTABLE**, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of TTBCR.T1SZ as follows:

- If TTBCR.T1SZ is 0 or 1, x = 5 - TTBCR.T1SZ.
- If TTBCR.T1SZ is greater than 1, x = 14 - TTBCR.T1SZ.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

This field resets to an architecturally **UNKNOWN** value.

**CnP, bit [0]**

**When ARMv8.2-TTCNP is implemented:**

Common not Private. When TTBCR.EAE == 1, this bit indicates whether each entry that is pointed to by TTBR1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1.
**CnP**

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0 | The translation table entries pointed to by this instance of TTBR1, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR1 for other PEs in the Inner Shareable domain. This is not affected by:  
  - The value of TTBR1.CnP on those other PEs.  
  - The value of TTBCR.EAE on those other PEs.  
  - The value of the current ASID or, for the Non-secure instance of TTBR1, the value of the current VMID. |
| 0b1 | The translation table entries pointed to by this instance of TTBR1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1 for this instance of TTBR1 and all of the following apply:  
  - The translation table entries are pointed to by this instance of TTBR1.  
  - The value of the applicable TTBCR.EAE field is 1.  
  - The ASID is the same as the current ASID.  
  - For the Non-secure instance of TTBR1, the VMID is the same as the current VMID. |

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

**Note**

If the value of the TTBR1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Accessing the TTBR1**

Accesses to this register use the following encodings:

\[
\text{MRC\{\langle c\rangle\}\{\langle q\rangle\} \langle coproc\rangle, \{\#\langle opc1\rangle, \langle Rt\rangle, \langle CRn\rangle, \langle CRm\rangle\}, \{\#\langle opc2\rangle\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            return TTBR1_S<31:0>;
        else
            return TTBR1_NS<31:0>;
        end if
    else
        return TTBR1<31:0>;
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS<31:0>;
    else
        return TTBR1<31:0>;
    end if
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR1_S<31:0>;
    else
        return TTBR1_NS<31:0>;
    end if
end if

MCR{<c>}{<q>}{<coproc>}, {#}{<opc1>}, <Rt>, <CRn>, <CRm>{, {#}{<opc2>}}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15DISABLE == HIGH then
        UNDEFINED;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15DISABLE2 == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);
        end if
    else
        TTBR1 = ZeroExtend(R[t]);
    end if
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = ZeroExtend(R[t]);
    else
        TTBR1 = ZeroExtend(R[t]);
    end if
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15DISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15DISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);
        end if
    end if
end if

MRRC<coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
   UNDEFINED;
elsif PSTATE.EL == EL1 then
   if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
      AArch32.TakeHypTrapException(0x04);
   elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
      AArch64.AArch32SystemAccessTrap(EL2, 0x04);
   elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
      AArch32.TakeHypTrapException(0x04);
   elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
      if SCR.NS == '0' then
         return TTBR1_S;
      else
         return TTBR1_NS;
      end;
   else
      return TTBR1;
   end;
elsif PSTATE.EL == EL2 then
   if HaveEL(EL3) && ELUsingAArch32(EL3) then
      return TTBR1_NS;
   else
      return TTBR1;
   end;
elsif PSTATE.EL == EL3 then
   if SCR.NS == '0' then
      return TTBR1_S;
   else
      return TTBR1_NS;
   end;
end;

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];
        end
    else
        TTBR1 = R[t2]:R[t];
    end
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = R[t2]:R[t];
    else
        TTBR1 = R[t2]:R[t];
    end
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];
        end
    end

The VBAR characteristics are:

**Purpose**

When high exception vectors are not selected, holds the vector base address for exceptions that are not taken to Monitor mode or to Hyp mode.

Software must program VBAR(NS) with the required initial value as part of the PE boot sequence.

**Configuration**

AArch32 System register VBAR bits [31:0] are architecturally mapped to AArch64 System register VBAR_EL1[31:0].

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 they apply only to the Secure instance of the register. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VBAR is a 32-bit register.

**Field descriptions**

The VBAR bit assignments are:

![Vector Base Address Register](image)

**Bits [31:5]**

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

This field resets to an IMPLEMENTATION DEFINED value.

**Bits [4:0]**

Reserved, RES0.

**Accessing the VBAR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      return VBAR_S;
    else
      return VBAR_NS;
    else
      return VBAR;
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
      return VBAR_NS;
    else
      return VBAR;
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      return VBAR_S;
    else
      return VBAR_NS;
else
  return VBAR_NS;

MCR{<coproc>}{<q>}{<coproc>}, {#<opc1>}, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE == HIGH then
    UNDEFINED;
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.NS == '0' && CP15SDISABLE2 == HIGH then
    UNDEFINED;
  elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if SCR.NS == '0' then
      VBAR_S = R[t];
    else
      VBAR_NS = R[t];
    else
      VBAR = R[t];
  elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
      VBAR_NS = R[t];
    else
      VBAR = R[t];
  elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
      UNDEFINED;
  elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
    UNDEFINED;
  else
    if SCR.NS == '0' then
      VBAR_S = R[t];
    else
      VBAR_NS = R[t];

VBAR, Vector Base Address Register
VDFSR, Virtual SError Exception Syndrome Register

The VDFSR characteristics are:

**Purpose**

Provides the syndrome value reported to software on taking a virtual SError interrupt exception to EL1, or on executing an ESB instruction at EL1.

When a virtual SError interrupt is taken, the syndrome value is reported in DFSR.AET, ExT and the remainder of the DFSR is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

If the virtual SError interrupt is deferred by an ESB instruction, then the syndrome value is written to VDISR.

**Configuration**

AArch32 System register VDFSR bits [31:0] are architecturally mapped to AArch64 System register VSESR_EL2[31:0] when the highest implemented Exception level is using AArch64.

This register is present only when RAS is implemented. Otherwise, direct accesses to VDFSR are UNDEFINED.

If EL2 is not implemented, then VDFSR is RES0 from Monitor mode when SCR.NS == 1.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VDFSR is a 32-bit register.

**Field descriptions**

The VDFSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | AET | RES0 | ExT | RES0 |

**Bits [31:16]**

Reserved, RES0.

**AET, bits [15:14]**

When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[15:4] is set to VDFSR.AET.

When a virtual SError interrupt is deferred by an ESB instruction, VDISR[15:4] is set to VDFSR.AET.

This field resets to an architecturally UNKNOWN value.

**Bit [13]**

Reserved, RES0.

**ExT, bit [12]**

When a virtual SError interrupt is taken to EL1 using AArch32, DFSR[12] is set to VDFSR.ExT.

When a virtual SError interrupt is deferred by an ESB instruction, VDISR[12] is set to VDFSR.ExT.
This field resets to an architecturally **UNKNOWN** value.

**Bits [11:0]**

Reserved, **RES0**.

**Accessing the VDFSR**

Direct reads and writes of VDFSR are **UNDEFINED** if EL3 is implemented and using AArch32 in all Secure privileged modes other than Monitor mode.

If EL2 is not implemented, then VDFSR is **RES0** from Monitor mode when **SCR.NS == 1**.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}\ <\text{coproc},\ {#}\<\text{opc1},\ <\text{Rt},\ <\text{CRn},\ <\text{CRm}\{,\ {#}\<\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b11</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
else
    PSTATE.EL == EL2 then
        return VDFSR;
elif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
elif PSTATE.EL == EL3 then
        return VDFSR;

\[
\text{MCR}\{<c>\}{<q>\ <\text{coproc},\ {#}\<\text{opc1},\ <\text{Rt},\ <\text{CRn},\ <\text{CRm}\{,\ {#}\<\text{opc2}\}}
\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b111</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b11</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
else
    PSTATE.EL == EL2 then
        VDFSR = R[t];
elif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
elif PSTATE.EL == EL3 then
        VDFSR = R[t];
VDISR, Virtual Deferred Interrupt Status Register

The VDISR characteristics are:

**Purpose**

Records that an SError interrupt has been consumed by an ESB instruction.

**Configuration**

AArch32 System register VDISR bits [31:0] are architecturally mapped to AArch64 System register VDISR_EL2[31:0]. This register is present only when RAS is implemented. Otherwise, direct accesses to VDISR are UNDEFINED.

If EL2 is not implemented, then VDISR is RES0 from Monitor mode when SCR.NS == 1.

RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VDISR is a 32-bit register.

**Field descriptions**

The VDISR bit assignments are:

When TTBCR.EAE == 0:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>RES0</td>
<td>AET</td>
<td>RES0</td>
<td>ExT</td>
<td>RES0</td>
<td>FS[4]</td>
<td>LPAE</td>
<td>RES0</td>
<td>RES0</td>
<td>FS[3:0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**A, bit [31]**

Set to 1 when an ESB instruction defers a virtual SError interrupt.

This field resets to an architecturally UNKNOWN value.

**Bits [30:16]**

Reserved, RES0.

**AET, bits [15:14]**

The value copied from VDFSR.AET.

This field resets to an architecturally UNKNOWN value.

**Bit [13]**

Reserved, RES0.

**ExT, bit [12]**

The value copied from VDFSR.ExT.

This field resets to an architecturally UNKNOWN value.
Bit [11]

Reserved, RES0.

FS[4], bit [10]

This field is bit[4] of FS[4:0].

Fault status code. Set to 0b10110 when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>FS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10110</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.

The FS field is split as follows:

- FS[4] is VDISR[10].
- FS[3:0] is VDISR[3:0].

This field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to TTBCR.EAE when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Using the Short-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

FS[3:0], bits [3:0]

This field is bits[3:0] of FS[4:0].


When TTBCR.EAE == 1:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|-----------------------------------------------|-----------------------------------------------|
| A | RES0 | AET | RES0 | Ext | RES0 | LPAE | RES0 | STATUS |

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

This field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from VDFSR AET.
This field resets to an architecturally **UNKNOWN** value.

**Bit [13]**

Reserved, **RES0**.

**ExT, bit [12]**

The value copied from **VDFSR** ExT.

This field resets to an architecturally **UNKNOWN** value.

**Bits [11:10]**

Reserved, **RES0**.

**LPAE, bit [9]**

Format.

Set to **TTBCR.EAE** when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>LPAE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Using the Long-descriptor translation table format.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [8:6]**

Reserved, **RES0**.

**STATUS, bits [5:0]**

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError interrupt.

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b010001</td>
<td>Asynchronous SError interrupt.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the VDISR**

Direct reads and writes of VDFS are **UNDEFINED** if EL3 is implemented and using AArch32 in all Secure privileged modes other than Monitor mode.

An indirect write to VDISR made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of **DISR** occurring in program order after the ESB instruction.

If EL2 is not implemented, then VDISR is **RES0** from Monitor mode when **SCR.NS == 1**.

Accesses to this register use the following encodings:

\[
\text{MRC}\{<c>\}{<q>}\ <\text{coproc}, \#<\text{opc1}, <\text{Rt}, <\text{CRn}, <\text{CRm}\{, \#<\text{opc2}\}}\]

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return VDISR;
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    return VDISR;
else
  return VDISR;
end if;

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  VDISR = R[t];
elsif PSTATE.EL == EL3 then
  if SCR.NS == '0' then
    UNDEFINED;
  else
    VDISR = R[t];
else
  return VDISR;
end if;

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b001</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
    return VDISR_EL2;
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
    return VDISR;
  else
    return DISR;
elsif PSTATE.EL == EL2 then
  return DISR;
elsif PSTATE.EL == EL3 then
  return DISR;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
    VDISR_EL2 = R[t];
  elsif EL2Enabled() && HCR.AMO == '1' then
    VDISR = R[t];
  else
    DISR = R[t];
elsif PSTATE.EL == EL2 then
  DISR = R[t];
else
  DISR = R[t];
The VMPIDR characteristics are:

**Purpose**

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by Non-secure EL1 reads of MPIDR.

**Configuration**

AArch32 System register VMPIDR bits [31:0] are architecturally mapped to AArch64 System register VMPIDR_EL2[31:0].

If EL2 is not implemented but EL3 is implemented, this register takes the value of the MPIDR.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VMPIDR is a 32-bit register.

**Field descriptions**

The VMPIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>U</td>
<td>RES0</td>
<td>MT</td>
<td>Aff2</td>
<td>Aff1</td>
<td>Aff0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**M, bit [31]**

Indicates whether this implementation includes the functionality introduced by the ARMv7 Multiprocessing Extensions. The possible values of this bit are:

<table>
<thead>
<tr>
<th>M</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This implementation does not include the ARMv7 Multiprocessing Extensions functionality.</td>
</tr>
<tr>
<td>0b1</td>
<td>This implementation includes the ARMv7 Multiprocessing Extensions functionality.</td>
</tr>
</tbody>
</table>

From Armv8 this bit is RES1.

**U, bit [30]**

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system. The possible values of this bit are:

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Processor is part of a multiprocessor system.</td>
</tr>
<tr>
<td>0b1</td>
<td>Processor is part of a uniprocessor system.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to the value in MPIDR.U.

**Bits [29:25]**

Reserved, RES0.
MT, bit [24]
Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels. The possible values of this bit are:

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Performance of PEs at the lowest affinity level is largely independent.</td>
</tr>
<tr>
<td>0b1</td>
<td>Performance of PEs at the lowest affinity level is very interdependent.</td>
</tr>
</tbody>
</table>

In a system where the PE resets into EL2 or EL3, this field resets to the value in MPIDR.MT.

Aff2, bits [23:16]
Affinity level 2. See the description of Aff0 for more information.
In a system where the PE resets into EL2 or EL3, this field resets to the value in MPIDR.Aff2.

Aff1, bits [15:8]
Affinity level 1. See the description of Aff0 for more information.
In a system where the PE resets into EL2 or EL3, this field resets to the value in MPIDR.Aff1.

Aff0, bits [7:0]
Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.Aff2, Aff1, Aff0} or MPIDR_EL1.Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.
In a system where the PE resets into EL2 or EL3, this field resets to the value in MPIDR.Aff0.

Accessing the VMPIDR
Accesses to this register use the following encodings:

```
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
UNDEFINED;
elsf PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return VMPIDR;
elsif PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    return MPIDR;
  elsif SCR.NS == '0' then
    UNDEFINED;
elsf else
    return VMPIDR;
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  VMPIDR = R[t];
else if PSTATE.EL == EL3 then
  if !HaveEL(EL2) then
    //no operation
  elsif SCR.NS == '0' then
    UNDEFINED;
  else
    VMPIDR = R[t];
else
  VMPIDR = R[t];

if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
    AArch32.TakeHypTrapException(0x03);
  elsif EL2Enabled() && !ELUsingAArch32(EL2) then
    return VMPIDR_EL2<31:0>;
  elsif EL2Enabled() && ELUsingAArch32(EL2) then
    return VMPIDR;
  else
    return MPIDR;
else if PSTATE.EL == EL2 then
  return MPIDR;
else if PSTATE.EL == EL3 then
  return MPIDR;
VPIDR, Virtualization Processor ID Register

The VPIDR characteristics are:

**Purpose**

Holds the value of the Virtualization Processor ID. This is the value returned by Non-secure EL1 reads of MIDR.

**Configuration**

AArch32 System register VPIDR bits [31:0] are architecturally mapped to AArch64 System register VPIDR_EL2[31:0].

If EL2 is not implemented but EL3 is implemented, this register takes the value of the MIDR.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VPIDR is a 32-bit register.

**Field descriptions**

The VPIDR bit assignments are:

<table>
<thead>
<tr>
<th>Implementer</th>
<th>Variant</th>
<th>Architecture</th>
<th>PartNum</th>
<th>Revision</th>
</tr>
</thead>
</table>

**Implementer, bits [31:24]**

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

<table>
<thead>
<tr>
<th>Hex representation</th>
<th>ASCII representation</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x41</td>
<td>A</td>
<td>Arm Limited</td>
</tr>
<tr>
<td>0x42</td>
<td>B</td>
<td>Broadcom Corporation</td>
</tr>
<tr>
<td>0x43</td>
<td>C</td>
<td>Cavium Inc.</td>
</tr>
<tr>
<td>0x44</td>
<td>D</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>0x49</td>
<td>I</td>
<td>Infineon Technologies AG</td>
</tr>
<tr>
<td>0x40</td>
<td>M</td>
<td>Motorola or Freescale</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Semiconductor Inc.</td>
</tr>
<tr>
<td>0x4E</td>
<td>N</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>0x58</td>
<td>P</td>
<td>Applied Micro Circuits Corporation</td>
</tr>
<tr>
<td>0x51</td>
<td>Q</td>
<td>Qualcomm Inc.</td>
</tr>
<tr>
<td>0x56</td>
<td>V</td>
<td>Marvell International Ltd.</td>
</tr>
<tr>
<td>0x69</td>
<td>i</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

In a system where the PE resets into EL2 or EL3, this field resets to the value in MIDR, Implementer.

**Variant, bits [23:20]**

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.
In a system where the PE resets into EL2 or EL3, this field resets to the value in MIDR.Variant.

**Architecture, bits [19:16]**

The permitted values of this field are:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Armv4.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Armv4T.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Armv5 (obsolete).</td>
</tr>
<tr>
<td>0b0100</td>
<td>Armv5T.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Armv5TE.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Armv5TEJ.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv6.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Architectural features are individually identified in the ID_* registers, see ‘ID registers’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K12.5.3.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In a system where the PE resets into EL2 or EL3, this field resets to the value in MIDR.Architecture.

**PartNum, bits [15:4]**

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

In a system where the PE resets into EL2 or EL3, this field resets to the value in MIDR.PartNum.

**Revision, bits [3:0]**

An IMPLEMENTATION DEFINED revision number for the device.

In a system where the PE resets into EL2 or EL3, this field resets to the value in MIDR.Revision.

**Accessing the VPIDR**

Accesses to this register use the following encodings:

```
MRC{<c>{<q>}} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}
```

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    return VPIDR;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return MIDR;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        VPIDR = R[t];
    end
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
    end
elsif PSTATE.EL == EL2 then
    return VPIDR;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        //no operation
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        VPIDR = R[t];
    end
MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VPIDR;
    else
        return MIDR;
else
    return MIDR;
elsif PSTATE.EL == EL2 then
    return MIDR;
elsif PSTATE.EL == EL3 then
    return MIDR;
VTCR, Virtualization Translation Control Register

The VTCR characteristics are:

**Purpose**

The control register for stage 2 of the Non-secure PL1&0 translation regime.

**Note**

This stage of translation always uses the Long-descriptor translation table format.

**Configuration**

AArch32 System register VTCR bits [31:0] are architecturally mapped to AArch64 System register `VTCR_EL2[31:0]`.

If EL2 is not implemented, this register is `RES0` from EL3.

RW fields in this register reset to architecturally `UNKNOWN` values.

**Attributes**

VTCR is a 32-bit register.

**Field descriptions**

The VTCR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES1</td>
<td>RES0</td>
<td>HWU62</td>
<td>HWU61</td>
<td>HWU60</td>
<td>HWU59</td>
<td>RES0</td>
<td>SH0</td>
<td>ORGN0</td>
<td>RGNO</td>
<td>SL0</td>
<td>RES0</td>
<td>S</td>
<td>T0SZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bit [31]**

Reserved, RES1.

**Bits [30:29]**

Reserved, RES0.

**HWU62, bit [28]**

When ARMv8.2-TTPBHA is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU62</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally `UNKNOWN` value.
Otherwise:

Reserved, RES0.

**HWU61, bit [27]**

**When ARMv8.2-TTPBHA is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU61</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU60, bit [26]**

**When ARMv8.2-TTPBHA is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU60</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HWU59, bit [25]**

**When ARMv8.2-TTPBHA is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

<table>
<thead>
<tr>
<th>HWU59</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RES0.

Bits [24:14]
Reserved, RES0.

SH0, bits [13:12]
Shareability attribute for memory associated with translation table walks using VTTBR.

<table>
<thead>
<tr>
<th>SH0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Inner Shareable.</td>
</tr>
</tbody>
</table>

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Unallocated values in fields of AArch32 System registers and translation table entries' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section K1.1.11.

This field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]
Outer cacheability attribute for memory associated with translation table walks using VTTBR.

<table>
<thead>
<tr>
<th>ORGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]
Inner cacheability attribute for memory associated with translation table walks using VTTBR.

<table>
<thead>
<tr>
<th>IRGN0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Normal memory, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]
Starting level for translation table walks using VTTBR.

<table>
<thead>
<tr>
<th>SL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Start at level 2</td>
</tr>
<tr>
<td>0b01</td>
<td>Start at level 1</td>
</tr>
</tbody>
</table>

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of T0SZ, then a stage 2 level 1 Translation fault is generated.
This field resets to an architecturally **UNKNOWN** value.

**Bit [5]**

Reserved, RES0.

**S, bit [4]**

Sign extension bit. This bit must be programmed to the value of T0SZ[3]. If it is not, then the behavior is **CONSTRAINED UNPREDICTABLE** and the stage 2 T0SZ value is treated as an **UNKNOWN** value, see 'Misprogramming VTCR.S' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to an architecturally **UNKNOWN** value.

**T0SZ, bits [3:0]**

The size offset of the memory region addressed by VTTBR. The region size is $2^{(32-T0SZ)}$ bytes.

This field holds a four-bit signed integer value, meaning it supports values from -8 to 7.

---

**Note**

This is different from the other translation control registers, where TnSZ holds a three-bit unsigned integer, supporting values from 0 to 7.

---

If this field is programmed to a value that is not consistent with the programming of SL0 then a stage 2 level 1 Translation fault is generated.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the VTCR**

Accesses to this register use the following encodings:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>

```plaintext
if PSTATE.EL == EL0 then
  UNDEFINED;
elsif PSTATE.EL == EL1 then
  if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
  elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
    AArch32.TakeHypTrapException(0x03);
  else
    UNDEFINED;
elsif PSTATE.EL == EL2 then
  return VTCR;
else
  PSTATE.EL == EL3 then
    if SCR.NS == '0' then
      UNDEFINED;
    else
      return VTCR;
  end if
MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
</tr>
</tbody>
</table>
```

Page 3095
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VTCR = R[t];
VTTBR, Virtualization Translation Table Base Register

The VTTBR characteristics are:

**Purpose**

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the Non-secure PL1&0 translation regime, and other information for this translation regime.

**Configuration**

AArch32 System register VTTBR bits [63:0] are architecturally mapped to AArch64 System register VTTBR_EL2[63:0].

If EL2 is not implemented, this register is RES0 from EL3.

Some or all RW fields of this register have defined reset values. These apply only if the PE resets into EL2 with EL2 using AArch32, or into EL3 with EL3 using AArch32. Otherwise, RW fields in this register reset to architecturally UNKNOWN values.

**Attributes**

VTTBR is a 64-bit register.

**Field descriptions**

The VTTBR bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:56]</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
</tr>
<tr>
<td>63  62  61  60  59  58  57  56  55  54  53  52  51  50  49  48  47  46  45  44  43  42  41  40  39  38  37  36  35  34  33  32</td>
</tr>
<tr>
<td>31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9   8   7   6   5   4   3   2   1   0</td>
</tr>
</tbody>
</table>

**Bits [63:56]**

Reserved, RES0.

**VMID, bits [55:48]**

The VMID for the translation table.

In a system where the PE resets into EL2 or EL3, this field resets to 0.

**BADDR, bits [47:1]**

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of VTCR.SL0 and VTCR.T0SZ as follows:

- If VTCR.SL0 is 0b00, meaning that lookup starts at level 2, then x is 14 - VTCR.T0SZ.
- If VTCR.SL0 is 0b01, meaning that lookup starts at level 1, then x is 5 - VTCR.T0SZ.
- If VTCR.SL0 is either 0b10 or 0b11 then a stage 2 level 1 Translation fault is generated.
If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

In a system where the PE resets into EL2 or EL3, this field resets to an architecturally **UNKNOWN** value.

**CnP, bit [0]**

**When ARMv8.2-TTCNP is implemented:**

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR.CnP is 1.

<table>
<thead>
<tr>
<th>CnP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The translation table entries pointed to by VTTBR are permitted to differ from the entries for VTTBR for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID.</td>
</tr>
<tr>
<td>0b1</td>
<td>The translation table entries pointed to by VTTBR are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR.CnP is 1 and the VMID is the same as the current VMID.</td>
</tr>
</tbody>
</table>

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

**Note**

If the value of the VTTBR.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBRs do not point to the same translation table entries when the VMID value is the same as the current VMID, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see ‘**CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values**’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

In a system where the PE resets into EL2 or EL3, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**Accessing the VTTBR**

Accesses to this register use the following encodings:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0110</td>
</tr>
</tbody>
</table>
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTTBR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return VTTBR;
end if

<table>
<thead>
<tr>
<th>coproc</th>
<th>CRm</th>
<th>opcl</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0110</td>
</tr>
</tbody>
</table>

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTTBR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VTTBR = R[t2]:R[t];
    end if

System Register index by instruction and encoding

Below are indexes for registers and operations accessed in the following ways:

For AArch32

- MCR/MRC
- MCRR/MRRC
- MRS/MSR
- VMRS/VMSR

For AArch64

- AT
- CFP
- CPP
- DC
- DVP
- IC
- MRS/MSR
- TLBI

Registers and operations in AArch32

Accessed using MCR/MRC:

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>DBGIDR</td>
<td>Debug ID Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
<td>DBGDTRXext</td>
<td>Debug OS Lock Data Transfer Register, Receive, External View</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b001</td>
<td>0b000</td>
<td>DBGDSCRint</td>
<td>Debug Status and Control Register, Internal View</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
<td>0b000</td>
<td>DBGDCCINT</td>
<td>DCC Interrupt Enable Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
<td>0b010</td>
<td>DBGDSCRext</td>
<td>Debug Status and Control Register, External View</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b011</td>
<td>0b010</td>
<td>DBGDTRXext</td>
<td>Debug OS Lock Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
<td>0b000</td>
<td>DBGDTRXint</td>
<td>Debug Data Transfer Register, Receive</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b010</td>
<td>0b000</td>
<td>DBGDTRXint</td>
<td>Debug Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b011</td>
<td>0b000</td>
<td>DBGWFAR</td>
<td>Debug Watchpoint Fault Address Register</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b010</td>
<td>DBGOSECCR</td>
<td>Debug OS Lock Catch Control Registers</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b000</td>
<td>DBGVCR</td>
<td>Debug Vector Catch Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>n[3:0]</td>
<td>0b100</td>
<td>DBGBVR&lt;n&gt;</td>
<td>Debug Breakpoint Value Registers</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>n[3:0]</td>
<td>0b101</td>
<td>DBGBCR&lt;n&gt;</td>
<td>Debug Breakpoint Control Registers</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>n[3:0]</td>
<td>0b110</td>
<td>DBGWVR&lt;n&gt;</td>
<td>Debug Watchpoint Value Registers</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>n[3:0]</td>
<td>0b111</td>
<td>DBGWCR&lt;n&gt;</td>
<td>Debug Watchpoint Control Registers</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>DBGDRAR</td>
<td>Debug ROM Address Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b100</td>
<td>DBGOSLR</td>
<td>Debug OS Lock Access Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b100</td>
<td>DBGOSLSR</td>
<td>Debug OS Lock Status Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b001</td>
<td>0b100</td>
<td>DBGOSDLR</td>
<td>Debug OS Double Lock Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b100</td>
<td>DBGPRCR</td>
<td>Debug Power Control Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>n[3:0]</td>
<td>0b001</td>
<td>DBGBXVR&lt;n&gt;</td>
<td>Debug Breakpoint Extended Value Registers</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>DBGDSAR</td>
<td>Debug Self Address Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b000</td>
<td>0b111</td>
<td>DBGDEVID2</td>
<td>Debug Device ID register 2</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b000</td>
<td>0b111</td>
<td>DBGDEVID1</td>
<td>Debug Device ID register 1</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b001</td>
<td>0b111</td>
<td>DBGDEVID</td>
<td>Debug Device ID register 3</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b010</td>
<td>0b111</td>
<td>DBGCLAIMSET</td>
<td>Debug CLAIM Tag Set register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b1000</td>
<td>0b110</td>
<td>DBGCLAIMCLR</td>
<td>Debug CLAIM Tag Clear register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b011</td>
<td>0b1000</td>
<td>0b110</td>
<td>DBGAUTHSTATUS</td>
<td>Debug Authentication Status register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>JIDR</td>
<td>Jazelle ID Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>0b000</td>
<td>IOSCR</td>
<td>Jazelle OS Control Register</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b111 0b0010 0b0000 0b000</td>
<td>JMCR</td>
<td>Jazelle Main Configuration Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0000 0b000</td>
<td>MIDR</td>
<td>Main ID Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0000 0b001</td>
<td>CTR</td>
<td>Cache Type Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0000 0b010</td>
<td>TCMTR</td>
<td>TCM Type Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0000 0b011</td>
<td>TLBTR</td>
<td>TLB Type Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0000 0b101</td>
<td>MPIR</td>
<td>Multiprocessor Affinity Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0000 0b110</td>
<td>REVIDR</td>
<td>Revision ID Register</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b000</td>
<td>ID_PFR0</td>
<td>Processor Feature Register 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b001</td>
<td>ID_PFR1</td>
<td>Processor Feature Register 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b010</td>
<td>ID_DFR0</td>
<td>Debug Feature Register 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b011</td>
<td>ID_AFR0</td>
<td>Auxiliary Feature Register 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b100</td>
<td>ID_MMFR0</td>
<td>Memory Model Feature Register 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b101</td>
<td>ID_MMFR1</td>
<td>Memory Model Feature Register 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b110</td>
<td>ID_MMFR2</td>
<td>Memory Model Feature Register 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0001 0b111</td>
<td>ID_MMFR3</td>
<td>Memory Model Feature Register 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b000</td>
<td>ID_ISAR0</td>
<td>Instruction Set Attribute Register 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b001</td>
<td>ID_ISAR1</td>
<td>Instruction Set Attribute Register 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b010</td>
<td>ID_ISAR2</td>
<td>Instruction Set Attribute Register 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b011</td>
<td>ID_ISAR3</td>
<td>Instruction Set Attribute Register 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b100</td>
<td>ID_ISAR4</td>
<td>Instruction Set Attribute Register 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b101</td>
<td>ID_ISAR5</td>
<td>Instruction Set Attribute Register 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000 0b0000 0b0010 0b110</td>
<td>ID_MMFR4</td>
<td>Memory Model Feature Register 4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>OPC</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b111</td>
<td>ID_ISAR6</td>
<td>Instruction Set Attribute Register 6</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b100</td>
<td>ID_PFR2</td>
<td>Processor Feature Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b101</td>
<td>ID_DFR1</td>
<td>Debug Feature Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b110</td>
<td>ID_MMFR5</td>
<td>Memory Mode Feature Register 5</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
<td>SCTLR</td>
<td>System Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
<td>ACTLR</td>
<td>Auxiliary Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b010</td>
<td>CPACR</td>
<td>Architectural Feature Access Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b011</td>
<td>ACTLR2</td>
<td>Auxiliary Control Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
<td>SCR</td>
<td>Secure Configuration Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
<td>SDER</td>
<td>Secure Debug Enable Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
<td>NSACR</td>
<td>Non-Secure Access Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
<td>TRFCR</td>
<td>Trace Filter Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
<td>SDCR</td>
<td>Secure Debug Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
<td>TTBR0</td>
<td>Translation Table Base Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
<td>TTBR1</td>
<td>Translation Table Base Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
<td>TTBCR</td>
<td>Translation Table Base Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b111</td>
<td>TTBCR2</td>
<td>Translation Table Base Control Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b000</td>
<td>DACR</td>
<td>Domain Access Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0110</td>
<td>0b000</td>
<td>ICC_PMR</td>
<td>Interrupt Controller Interrupt Priority Mask Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b000</td>
<td>DFSR</td>
<td>Data Fault Status Register</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>-------------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b001</td>
<td>IFSR</td>
<td>Instruction Fault Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
<td>ADFSR</td>
<td>Auxiliary Data Fault Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
<td>AIFSР</td>
<td>Auxiliary Instruction Fault Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
<td>ERRIDR</td>
<td>Error Record Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b001</td>
<td>ERBSELR</td>
<td>Error Record Select Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b000</td>
<td>ERXFR</td>
<td>Selected Error Record Feature Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b001</td>
<td>ERXCTLR</td>
<td>Selected Error Record Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b010</td>
<td>ERXSTATUS</td>
<td>Selected Error Record Primary Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b011</td>
<td>ERXADDR</td>
<td>Selected Error Record Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b100</td>
<td>ERXFR2</td>
<td>Selected Error Record Feature Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b101</td>
<td>ERXCTLR2</td>
<td>Selected Error Record Control Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b111</td>
<td>ERXADDR2</td>
<td>Selected Error Record Address Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b000</td>
<td>ERXMISC0</td>
<td>Selected Error Record Miscellaneous Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b001</td>
<td>ERXMISC1</td>
<td>Selected Error Record Miscellaneous Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b010</td>
<td>ERXMISC2</td>
<td>Selected Error Record Miscellaneous Register 4</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b011</td>
<td>ERXMISC3</td>
<td>Selected Error Record Miscellaneous Register 5</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b100</td>
<td>ERXMISC2</td>
<td>Selected Error Record Miscellaneous Register 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b101</td>
<td>ERXMISC3</td>
<td>Selected Error Record Miscellaneous Register 3</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b110</td>
<td>ERXMISC6</td>
<td>Selected Error Record Register</td>
</tr>
</tbody>
</table>

System Register index by instruction and encoding
### Register selectors

<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b111</td>
<td>ERXMISC7</td>
<td>Selected Error Record Miscellaneous Register 6</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
<td>DFAR</td>
<td>Data Fault Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b010</td>
<td>IFAR</td>
<td>Instruction Fault Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0001</td>
<td>0b000</td>
<td>ICIALLUIS</td>
<td>Instruction Cache Invalidate All PoU, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0001</td>
<td>0b110</td>
<td>BPIALLIS</td>
<td>Branch Predictor Invalidate All, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b100</td>
<td>CFPRCTX</td>
<td>Control Flow Prediction Restriction by Context</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b101</td>
<td>DVPRCTX</td>
<td>Data Value Prediction Restriction by Context</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b111</td>
<td>CPPRCTX</td>
<td>Cache Prefetch Prediction Restriction by Context</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b000</td>
<td>PAR</td>
<td>Physical Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b000</td>
<td>ICIALLU</td>
<td>Instruction Cache Invalidate All PoU</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b001</td>
<td>ICIMVAU</td>
<td>Instruction Cache line Invalidate by VA to PoU</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b100</td>
<td>CP15ISB</td>
<td>Instruction Synchronization Barrier System instruction</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b110</td>
<td>BPIALL</td>
<td>Branch Predictor Invalidate All</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b111</td>
<td>BPIMVA</td>
<td>Branch Predictor Invalidate by VA</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b001</td>
<td>DCIMVAC</td>
<td>Data Cache line Invalidate by VA to PoC</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b010</td>
<td>DCISW</td>
<td>Data Cache line Invalidate by Set/way</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------------------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b000</td>
<td></td>
<td>ATS1CPR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b001</td>
<td></td>
<td>ATS1CPW</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b010</td>
<td></td>
<td>ATS1CUR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b011</td>
<td></td>
<td>ATS1CUW</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b100</td>
<td></td>
<td>ATS12NSOPR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b101</td>
<td></td>
<td>ATS12NSOPW</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b110</td>
<td></td>
<td>ATS12NSOUR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b111</td>
<td></td>
<td>ATS12NSOUW</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b101</td>
<td>0b000</td>
<td></td>
<td>ATS1CPRP</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b101</td>
<td>0b001</td>
<td></td>
<td>ATS1CPWP</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b101</td>
<td>0b001</td>
<td></td>
<td>DCCMVAC</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b101</td>
<td>0b010</td>
<td></td>
<td>DCCSW</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b101</td>
<td>0b100</td>
<td></td>
<td>CP15DSB</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b101</td>
<td>0b101</td>
<td></td>
<td>CP15DMB</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1011</td>
<td>0b001</td>
<td>DCCMVAU</td>
<td>Data Cache line Clean by VA to PoU</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b001</td>
<td>DCCIMVAC</td>
<td>Data Cache line Clean and Invalidate by VA to PoC</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b010</td>
<td>DCCISW</td>
<td>Data Cache line Clean and Invalidate by Set/Way</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>TLBIALLIS</td>
<td>TLB Invalidate All, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
<td>TLBIMVAIS</td>
<td>TLB Invalidate by VA, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b010</td>
<td>TLBIASIDIS</td>
<td>TLB Invalidate by ASID match, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b011</td>
<td>TLBIMVAIS</td>
<td>TLB Invalidate by VA, All ASID, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
<td>TLBIMVALIS</td>
<td>TLB Invalidate by VA, Last level, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b111</td>
<td>TLBIMVAALIS</td>
<td>TLB Invalidate by VA, All ASID, Last level, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b000</td>
<td>ITLBIALL</td>
<td>Instruction TLB Invalidate All</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
<td>ITLBMVA</td>
<td>Instruction TLB Invalidate by VA</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b010</td>
<td>ITLBIASID</td>
<td>Instruction TLB Invalidate by ASID match</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b011</td>
<td>ITLBIALL</td>
<td>Instruction TLB Invalidate by ASID match</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b000</td>
<td>DTLBIALL</td>
<td>Data TLB Invalidate All</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
<td>DTLBMVA</td>
<td>Data TLB Invalidate by VA</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b010</td>
<td>DTLBIASID</td>
<td>Data TLB Invalidate by ASID match</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>TLBIALL</td>
<td>TLB Invalidate All</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
<td>TLBMVA</td>
<td>TLB Invalidate by VA</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b010</td>
<td>TLBIASID</td>
<td>TLB Invalidate by ASID match</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b011</td>
<td>TLBMVAA</td>
<td>TLB Invalidate by VA, All ASID, Last level</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
<td>TLBMVAL</td>
<td>TLB Invalidate by VA, Last level</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b111</td>
<td>TLBMVAAL</td>
<td>TLB Invalidate by VA, All ASID, Last level</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b000</td>
<td>PMCR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b001</td>
<td>PMCntenSet</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b010</td>
<td>PMCntenCLR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b011</td>
<td>PMovSR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b100</td>
<td>PMswinc</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b101</td>
<td>Pmselector</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b110</td>
<td>Pmceid0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1100</td>
<td></td>
<td>0b111</td>
<td>Pmceid1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td></td>
<td>0b000</td>
<td>Pmccntr</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td></td>
<td>0b001</td>
<td>Pmxevtyper</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1101</td>
<td></td>
<td>0b010</td>
<td>Pmxevcntr</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td></td>
<td>0b000</td>
<td>Pmuserenr</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td></td>
<td>0b001</td>
<td>Pmintenset</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td></td>
<td>0b010</td>
<td>Pmintenclr</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td></td>
<td>0b011</td>
<td>PmoveSet</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>Register selectors CRn CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-----------------------------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001 0b1110 0b100</td>
<td></td>
<td></td>
<td>Status Set register</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001 0b1110 0b101</td>
<td></td>
<td></td>
<td>Performance Monitors Common Event Identification register 2</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1001 0b1110 0b110</td>
<td></td>
<td></td>
<td>Performance Monitors Common Event Identification register 3</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010 0b0011 0b000</td>
<td></td>
<td></td>
<td>PMMIR</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1010 0b0011 0b001</td>
<td></td>
<td></td>
<td>Auxiliary Memory Attribute Indirection Register 0</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b0000 0b000</td>
<td></td>
<td></td>
<td>Auxiliary Memory Attribute Indirection Register 1</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b0000 0b010</td>
<td></td>
<td></td>
<td>VBAR</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b0001 0b000</td>
<td></td>
<td></td>
<td>RMR</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b0001 0b001</td>
<td></td>
<td></td>
<td>ISR</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b1000 0b000</td>
<td></td>
<td></td>
<td>DISR</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b1000 0b000</td>
<td></td>
<td></td>
<td>ICC_IAR0</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b1000 0b001</td>
<td></td>
<td></td>
<td>ICC_EOIR0</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b1000 0b010</td>
<td></td>
<td></td>
<td>ICC_HPPIR0</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b1000 0b011</td>
<td></td>
<td></td>
<td>ICC_BPR0</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1100 0b1000 0b1:n[1:0]</td>
<td></td>
<td></td>
<td>ICC_APORn&gt;</td>
<td></td>
</tr>
</tbody>
</table>

System Register index by instruction and encoding
<table>
<thead>
<tr>
<th>coproc</th>
<th>opc1</th>
<th>CRn</th>
<th>CRm</th>
<th>opc2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
<td>ICC_APIR&lt;n&gt;</td>
<td>Interrupt Controller Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
<td>ICC_DIR</td>
<td>Interrupt Controller Deactivate Interrupt Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
<td>ICC_RPR</td>
<td>Interrupt Controller Running Priority Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b000</td>
<td>ICC_JAR1</td>
<td>Interrupt Controller Interrupt Acknowledge Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b001</td>
<td>ICC_EOFIR1</td>
<td>Interrupt Controller End Of Interrupt Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b010</td>
<td>ICC_HPPIR1</td>
<td>Interrupt Controller Highest Pending Interrupt Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
<td>ICC_BPR1</td>
<td>Interrupt Controller Binary Point Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
<td>ICC_CTLR</td>
<td>Interrupt Controller Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
<td>ICC_SRF</td>
<td>Interrupt Controller System Register Enable register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b110</td>
<td>ICC_IGRPEN0</td>
<td>Interrupt Controller Interrupt Group 0 Enable register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
<td>ICC_IGRPEN1</td>
<td>Interrupt Controller Interrupt Group 1 Enable register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
<td>FCSEIDR</td>
<td>FCSE Process ID register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
<td>CONTEXTIDR</td>
<td>Context ID Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
<td>TPIDRURW</td>
<td>PL0 Read/Write Software Thread ID Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b00</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b011</td>
<td>TPIDRUSO</td>
<td>PL0 Read-Only Software</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>Register selectors</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>--------------------</td>
<td>------</td>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b100</td>
<td>TPIDRPRW</td>
<td>Thread ID Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b000</td>
<td>AMCR</td>
<td>Activity Monitors Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b001</td>
<td>AMCFGR</td>
<td>Activity Monitors Configuration Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b010</td>
<td>AMCGCR</td>
<td>Activity Monitors Counter Group Configuration Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b011</td>
<td>AMUSERENR</td>
<td>Activity Monitors User Enable Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b100</td>
<td>AMCNTENCLR0</td>
<td>Activity Monitors Count Enable Clear Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b101</td>
<td>AMCNTENSET0</td>
<td>Activity Monitors Count Enable Set Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b000</td>
<td>AMCNTENCLR1</td>
<td>Activity Monitors Count Enable Clear Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b001</td>
<td>AMCNTENSET1</td>
<td>Activity Monitors Count Enable Set Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b011:n[3]</td>
<td>n[2:0]</td>
<td>AMEVTYPE0&lt;n&gt;</td>
<td>Activity Monitors Event Type Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b011:n[3]</td>
<td>n[2:0]</td>
<td>AMEVTYPE1&lt;n&gt;</td>
<td>Activity Monitors Event Type Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b000</td>
<td>CNTFRQ</td>
<td>Counter-timer Frequency register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
<td>CNTKCTL</td>
<td>Counter-timer Kernel Control register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
<td>CNTP_TVAL</td>
<td>Counter-timer Physical Time TimerValue register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
<td>CNTP_CTL</td>
<td>Counter-timer Physical Time Control register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
<td>CNTV_TVAL</td>
<td>Counter-timer Virtual Time TimerValue register</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>--------------------</td>
<td>--------</td>
<td>--------</td>
<td>------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
<td></td>
<td>CNTV_CTL</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b10:n[4:3]</td>
<td>n[2:0]</td>
<td></td>
<td>PMEVCNTR&lt;n&gt;</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b1111</td>
<td>0b111</td>
<td></td>
<td>PMCCFILTR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b11:n[4:3]</td>
<td>n[2:0]</td>
<td></td>
<td>PMEVTYPE&lt;n&gt;</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td></td>
<td>CCSIDR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
<td></td>
<td>CLIDR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
<td></td>
<td>CCSIDR2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
<td></td>
<td>AIDR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b010</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td></td>
<td>CSSELR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b000</td>
<td></td>
<td>DSPSR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b001</td>
<td></td>
<td>DLR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td></td>
<td>VPIDR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
<td></td>
<td>VMPIDR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
<td></td>
<td>HSCTRL</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
<td></td>
<td>HACTLR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b011</td>
<td></td>
<td>HACTLR2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
<td></td>
<td>HCR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
<td></td>
<td>HDCR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
<td></td>
<td>HCPTR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b011</td>
<td></td>
<td>HSTR</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
<td></td>
<td>HCR2</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>----------</td>
<td>--------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b111</td>
<td>HACR</td>
<td>Hyp Auxiliary Configuration Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
<td>HTRFCR</td>
<td>Hyp Trace Filter Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
<td>HTCR</td>
<td>Hyp Translation Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
<td>VTCR</td>
<td>Virtualization Translation Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b000</td>
<td>HADFSR</td>
<td>Hyp Auxiliary Data Fault Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b001</td>
<td>HAIFSR</td>
<td>Hyp Auxiliary Instruction Fault Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b010</td>
<td>0b000</td>
<td>HSR</td>
<td>Hyp Syndrome Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b011</td>
<td>0b000</td>
<td>VDFSR</td>
<td>Virtual SError Exception Syndrome Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
<td>HDFAR</td>
<td>Hyp Data Fault Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b010</td>
<td>HIFAR</td>
<td>Hyp Instruction Fault Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b100</td>
<td>HPFAR</td>
<td>Hyp IPA Fault Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b000</td>
<td>ATS1HR</td>
<td>Address Translate Stage 1 Hyp mode Read</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b001</td>
<td>ATS1HW</td>
<td>Address Translate Stage 1 Hyp mode Write</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b001</td>
<td>TLBHIPAS2IS</td>
<td>TLB Invalidated by Intermediate Physical Address, Stage 2, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b101</td>
<td>TLBHIPAS2LIS</td>
<td>TLB Invalidated by Intermediate Physical Address, Stage 2, Last level, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>TLBIALLHIS</td>
<td>TLB Invalidated All, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
<td>TLBIMVAHIS</td>
<td>TLB Invalidated by VA, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td>-------------------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b100</td>
<td>TLBIALLNSNHS</td>
<td>TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
<td>TLBIMVALHIS</td>
<td>TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b001</td>
<td>TLBHIPAS2</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0100</td>
<td>0b101</td>
<td>TLBHIPAS2L</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>TLBIALLH</td>
<td>TLB Invalidate All, Hyp mode</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
<td>TLBIMVAH</td>
<td>TLB Invalidate by VA, Hyp mode</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b100</td>
<td>TLBIALLNSNH</td>
<td>TLB Invalidate All, Non-Secure Non-Hyp</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
<td>TLBIMVALH</td>
<td>TLB Invalidate by VA, Last level, Hyp mode</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
<td>HMAIR0</td>
<td>Hyp Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b001</td>
<td>HMAIR1</td>
<td>Hyp Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
<td>HAMAIR0</td>
<td>Hyp Auxiliary Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b001</td>
<td>HAMAIR1</td>
<td>Hyp Auxiliary Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
<td>HVBAR</td>
<td>Hyp Vector Base Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
<td>HRMR</td>
<td>Hyp Reset Management Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
<td>VDISR</td>
<td>Virtual Deferred Interrupt Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b0:n[1:0]</td>
<td>ICH_AP0R&lt;n&gt;</td>
<td>Interrupt Controller Hyp Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>coproc</td>
<td>opc1</td>
<td>CRn</td>
<td>CRm</td>
<td>opc2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0(n[1:0])</td>
<td>ICH_AP1R(n&gt;)</td>
<td>Interrupt Controller Hyp Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b101</td>
<td>ICC_HSRE</td>
<td>Interrupt Controller Hyp System Register Enable Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b000</td>
<td>ICH_HCR</td>
<td>Interrupt Controller Hyp Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
<td>ICH_VTR</td>
<td>Interrupt Controller VGIC Type Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b010</td>
<td>ICH_MISR</td>
<td>Interrupt Controller Maintenance Interrupt Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
<td>ICH_EISR</td>
<td>Interrupt Controller End of Interrupt Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b101</td>
<td>ICH_ELRSR</td>
<td>Interrupt Controller Empty List Register Status Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b111</td>
<td>ICH_VMCR</td>
<td>Interrupt Controller Virtual Machine Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b110:n[3]</td>
<td>n[2:0]</td>
<td>ICH_LR(n&gt;)</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b111:n[3]</td>
<td>n[2:0]</td>
<td>ICH_LRC(n&gt;)</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
<td>HTPIDR</td>
<td>Hyp Software Thread ID Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
<td>CNTHCTL</td>
<td>Counter-timer Hyp Control register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
<td>CNTHP_TVAL</td>
<td>Counter-timer Hyp Physical Timer TimerValue register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
<td>CNTHP_CTL</td>
<td>Counter-timer Hyp Physical Timer Control register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
<td>ICC_MCTLR</td>
<td>Interrupt Controller Monitor Control Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
<td>ICC_MSRE</td>
<td>Interrupt Controller</td>
</tr>
</tbody>
</table>
### Accessed using MCRR/MRRC:

<table>
<thead>
<tr>
<th>coproc</th>
<th>Register selectors CRm</th>
<th>opc1</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1110</td>
<td>0b0001</td>
<td>0b0000</td>
<td>DBGDRAR</td>
<td>Debug ROM Address Register</td>
</tr>
<tr>
<td>0b1110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>DBGDSAR</td>
<td>Debug Self Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b000:n[3]</td>
<td>0b0:n[2:0]</td>
<td>AMEVCNTR0&lt;n&gt;</td>
<td>Activity Monitors Event Counter Registers 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0000</td>
<td>TTBRO</td>
<td>Translation Table Base Register 0</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0001</td>
<td>TTBRI</td>
<td>Translation Table Base Register 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0100</td>
<td>HTTBR</td>
<td>Hyp Translation Table Base Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b0010</td>
<td>0b0110</td>
<td>VTTBR</td>
<td>Virtualization Translation Table Base Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b010:n[3]</td>
<td>0b0:n[2:0]</td>
<td>AMEVCNTR1&lt;n&gt;</td>
<td>Activity Monitors Event Counter Registers 1</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b0111</td>
<td>0b0000</td>
<td>PAR</td>
<td>Physical Address Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1001</td>
<td>0b0000</td>
<td>PMCCNTR</td>
<td>Performance Monitors Cycle Count Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1100</td>
<td>0b0000</td>
<td>ICC_SGI1R</td>
<td>Interrupt Controller Software Generated Interrupt Group 1 Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1100</td>
<td>0b0001</td>
<td>ICC_ASGI1R</td>
<td>Interrupt Controller Alias Software Generated Interrupt Group 1 Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1100</td>
<td>0b0010</td>
<td>ICC_SGI0R</td>
<td>Interrupt Controller Software Generated Interrupt Group 0 Register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0000</td>
<td>CNTPCCT</td>
<td>Counter-timer Physical Count register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0001</td>
<td>CNTVCT</td>
<td>Counter-timer Virtual Count register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0010</td>
<td>CNTV_CVAL</td>
<td>Counter-timer Virtual Timer CompareValue register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0011</td>
<td>CNTV_CVAL</td>
<td>Counter-timer Virtual Timer CompareValue register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0100</td>
<td>CNTVOFF</td>
<td>Counter-timer Virtual Offset register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b0110</td>
<td>CNTHP_CVAL</td>
<td>Counter-timer Hyp Physical CompareValue register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b1000</td>
<td>CNTPCTSS</td>
<td>Counter-timer Self-Synchronized Physical Count register</td>
</tr>
<tr>
<td>0b1111</td>
<td>0b1110</td>
<td>0b1001</td>
<td>CNTVCTSS</td>
<td>Counter-timer Self-Synchronized Virtual Count register</td>
</tr>
</tbody>
</table>

### Accessed using MRS/MSR:

<table>
<thead>
<tr>
<th>Register selectors R M M1</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 0b1 0b1110</td>
<td>ELR_hyp</td>
<td>Exception Link Register (Hyp mode)</td>
</tr>
</tbody>
</table>
System Register index by instruction and encoding

### Register selectors

<table>
<thead>
<tr>
<th>R</th>
<th>M</th>
<th>M1</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>0b1110</td>
<td>SPSR_fiq</td>
<td>Saved Program Status Register (FIQ mode)</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0000</td>
<td>SPSR_irq</td>
<td>Saved Program Status Register (IRQ mode)</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0010</td>
<td>SPSR_svc</td>
<td>Saved Program Status Register (Supervisor mode)</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0100</td>
<td>SPSR_abt</td>
<td>Saved Program Status Register (Abort mode)</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b0110</td>
<td>SPSR_svc</td>
<td>Saved Program Status Register (Supervisor mode)</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b1100</td>
<td>SPSR_mon</td>
<td>Saved Program Status Register (Monitor mode)</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>0b1110</td>
<td>SPSR_hyp</td>
<td>Saved Program Status Register (Hyp mode)</td>
</tr>
</tbody>
</table>

#### Accessed using VMRS/VMSR:

<table>
<thead>
<tr>
<th>Register selectors</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>FPSID</td>
<td>Floating-Point System ID register</td>
</tr>
<tr>
<td>0b0001</td>
<td>FPSCR</td>
<td>Floating-Point Status and Control Register</td>
</tr>
<tr>
<td>0b0101</td>
<td>MVFR2</td>
<td>Media and VFP Feature Register 2</td>
</tr>
<tr>
<td>0b0110</td>
<td>MVFR1</td>
<td>Media and VFP Feature Register 1</td>
</tr>
<tr>
<td>0b0111</td>
<td>MVFR0</td>
<td>Media and VFP Feature Register 0</td>
</tr>
<tr>
<td>0b1000</td>
<td>FPEXC</td>
<td>Floating-Point Exception Control register</td>
</tr>
</tbody>
</table>

#### Registers and operations in AArch64

#### Accessed using AT:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
<td>AT</td>
<td>Address Translate Stage 1 EL1 Read</td>
</tr>
<tr>
<td>0b01</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
<td>AT</td>
<td>Address Translate Stage 1 EL1 Write</td>
</tr>
<tr>
<td>0b01</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b010</td>
<td>AT</td>
<td>Address Translate Stage 1 EL0 Read</td>
</tr>
<tr>
<td>0b01</td>
<td>0b00</td>
<td>0b1111</td>
<td>0b1000</td>
<td>0b011</td>
<td>AT</td>
<td>Address Translate Stage 1 EL0 Write</td>
</tr>
<tr>
<td>0b01</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b000</td>
<td>AT</td>
<td>Address Translate Stage 1 EL1 Read PAN</td>
</tr>
<tr>
<td>0b01</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b001</td>
<td>AT</td>
<td>Address Translate Stage 1 EL1 Write PAN</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
<td>AT</td>
<td>Address Translate Stage 1 EL2 Read</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
<td>AT</td>
<td>Address Translate Stage 1 EL2 Write</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b100</td>
<td>AT</td>
<td>Address Translate Stages 1 and 2 EL1 Read</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b101</td>
<td>AT</td>
<td>Address Translate Stages 1 and 2 EL1 Write</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
<td>AT</td>
<td>Address Translate Stages 1 and 2 EL0 Read</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b111</td>
<td>AT</td>
<td>Address Translate Stages 1 and 2 EL0 Write</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b000</td>
<td>AT</td>
<td>Address Translate Stage 1 EL3 Read</td>
</tr>
<tr>
<td>0b01</td>
<td>0b10</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b001</td>
<td>AT</td>
<td>Address Translate Stage 1 EL3 Write</td>
</tr>
</tbody>
</table>
### Accessed using CFP:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b100</td>
<td><strong>CFP</strong></td>
<td><strong>RCTX</strong> Control Flow Prediction Restriction by Context</td>
</tr>
</tbody>
</table>

### Accessed using CPP:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0011</td>
<td>0b111</td>
<td><strong>CPP</strong></td>
<td><strong>RCTX</strong> Cache Prefetch Prediction Restriction by Context</td>
</tr>
</tbody>
</table>

### Accessed using DC:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b001</td>
<td><strong>DC IVAC</strong></td>
<td>Data or unified Cache line Invalidate by VA to PoC</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b010</td>
<td><strong>DC ISW</strong></td>
<td>Data or unified Cache line Invalidate by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b011</td>
<td><strong>DC IGVAC</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b100</td>
<td><strong>DC IGSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b101</td>
<td><strong>DC IGDVAC</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0110</td>
<td>0b110</td>
<td><strong>DC IGDSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b010</td>
<td><strong>DC CSW</strong></td>
<td>Data or unified Cache line Clean by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b100</td>
<td><strong>DC CGSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b110</td>
<td><strong>DC CGDSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b010</td>
<td><strong>DC CISW</strong></td>
<td>Data or unified Cache line Clean and Invalidate by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b100</td>
<td><strong>DC CIGSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Clean and Invalidate of Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b110</td>
<td><strong>DC CIGDSW</strong></td>
<td>Data, Allocation Tag or unified Cache line Clean and Invalidate of Data and Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b001</td>
<td><strong>DC ZVA</strong></td>
<td>Data Cache Zero by VA</td>
</tr>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b011</td>
<td><strong>DC GVA</strong></td>
<td>Data Cache set Allocation Tag by VA</td>
</tr>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b100</td>
<td><strong>DC GZVA</strong></td>
<td>Data Cache set Allocation Tags and Zero by VA</td>
</tr>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b1010</td>
<td>0b001</td>
<td><strong>DC CVAC</strong></td>
<td>Data or unified Cache line Clean by VA to PoC</td>
</tr>
</tbody>
</table>
### Accessed using DVP:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b001</td>
<td>0b101</td>
<td>DVP</td>
<td>Data Value Prediction Restriction by Context</td>
</tr>
</tbody>
</table>

### Accessed using IC:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b1111</td>
<td>IC</td>
<td>Instruction Cache Invalidate All to PoU, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b000</td>
<td>0b1111</td>
<td>IC</td>
<td>Instruction Cache Invalidate All to PoU</td>
</tr>
<tr>
<td>0b01</td>
<td>0b011</td>
<td>0b0111</td>
<td>0b0101</td>
<td>0b001</td>
<td>-</td>
<td>IC</td>
<td>Instruction Cache line Invalidate by VA to PoU</td>
</tr>
</tbody>
</table>
### Accessed using MRS/MSR:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
<td>OSDTRRX_EL1</td>
<td>OS Lock Transfer Register, Receive</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
<td>MDCCINT_EL1</td>
<td>Monitor Interrupt Enable Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
<td>MDSCR_EL1</td>
<td>Monitor System Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
<td>OSDTRTX_EL1</td>
<td>OS Lock Transfer Register, Transmit</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b010</td>
<td>OSECCR_EL1</td>
<td>OS Lock Exception Catch Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b100</td>
<td>DBGVBR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Value Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b101</td>
<td>DBGWCR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b110</td>
<td>DBGWVR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Value Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0000</td>
<td>n[3:0]</td>
<td>0b111</td>
<td>DBGWCR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
<td>MDRAR_EL1</td>
<td>Monitor ROM Address Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b100</td>
<td>OSLAR_EL1</td>
<td>OS Lock Access Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
<td>OSLSR_EL1</td>
<td>OS Lock Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b100</td>
<td>OSDLR_EL1</td>
<td>OS Double Lock Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b1000</td>
<td>0b100</td>
<td>DBGPRCR_EL1</td>
<td>Debug PWR Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
<td>DBGCLAIMSET_EL1</td>
<td>Debug CLAIM Tag Set register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
<td>DBGCLAIMCLR_EL1</td>
<td>Debug CLAIM Tag Clear register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b00</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b110</td>
<td>DBGAUTHSTATUS_EL1</td>
<td>Debug Authentication Status register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
<td>TRCTRACEIDR</td>
<td>Trace ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
<td>TRCVICTLR</td>
<td>ViewInst Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b01</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b110</td>
<td>TRCIDR8</td>
<td>ID Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
<td>TRCIMSPEC0</td>
<td>IMP DEF Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
<td>TRCPRGCTLR</td>
<td>Program Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b001</td>
<td>TRCQCTRL</td>
<td>Q Element Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b010</td>
<td>TRCIVIECTLR</td>
<td>ViewInst Include/Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
<td>TRCIDR9</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
<td>TRCVISSCTRL</td>
<td>ViewInst Stop Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b000</td>
<td>TRCIDR10</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
<td>TRCSTATR</td>
<td>Trace Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
<td>TRCVIPCSSCTRL</td>
<td>ViewInst Stop PE Comparator Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b110</td>
<td>TRCIDR11</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b110</td>
<td>TRCIDR11</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000:n[1:0]</td>
<td>0b100</td>
<td>TRCEQEVVR&lt;n&gt;</td>
<td>Sequence State Trace Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000:n[1:0]</td>
<td>0b101</td>
<td>TRCCNTRLDVR&lt;n&gt;</td>
<td>Counter Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b000</td>
<td>TRCCONFIGR</td>
<td>Trace Configuration Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b110</td>
<td>TRCIDR12</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b110</td>
<td>TRCIDR13</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b000</td>
<td>TRCIAUXCTRL</td>
<td>Auxiliary Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b100</td>
<td>TRCEQORSTEVR</td>
<td>Sequence Reset Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b100</td>
<td>TRCEQOSTR</td>
<td>Sequence State Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b01:n[1:0]</td>
<td>0b101</td>
<td>TRCCNCTL&lt;n&gt;</td>
<td>Counter Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0:n[2:0]</td>
<td>0b111</td>
<td>TRCIMSPEC&lt;n&gt;</td>
<td>IMP DEF Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1000</td>
<td>0b000</td>
<td>TRCEVENTCTRL0R</td>
<td>Event Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1000</td>
<td>0b111</td>
<td>TRCIDR0</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1001</td>
<td>0b000</td>
<td>TRCEVENTCTRL1R</td>
<td>Event Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1001</td>
<td>0b111</td>
<td>TRCIDR1</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1010</td>
<td>0b000</td>
<td>TRCRSR</td>
<td>Resources Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1010</td>
<td>0b111</td>
<td>TRCIDR2</td>
<td>ID Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1011</td>
<td>0b000</td>
<td>TRCSTALLCTRLR</td>
<td>Stall Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1011</td>
<td>0b111</td>
<td>TRCICDR3</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b10:0b10:n[1:0]</td>
<td>0b100</td>
<td>TRCEXTINSEL&lt;n&gt;</td>
<td>External Select Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b10:0b10:n[1:0]</td>
<td>0b101</td>
<td>TRCCNTVR&lt;n&gt;</td>
<td>Counter Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1100</td>
<td>0b000</td>
<td>TRCTSCTRLR</td>
<td>Timestamp Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1100</td>
<td>0b111</td>
<td>TRCIDR4</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b000</td>
<td>TRCSYNCR</td>
<td>Synchronization Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1101</td>
<td>0b111</td>
<td>TRCCDCTRLR</td>
<td>Cycle Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1110</td>
<td>0b111</td>
<td>TRCIDR6</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1111</td>
<td>0b000</td>
<td>TRCBBCCTRLR</td>
<td>Branch Broadcast Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b1111</td>
<td>0b111</td>
<td>TRCIDR7</td>
<td>ID Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
<td>TRCOSLSR</td>
<td>Trace OS Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b0:n[2:0]</td>
<td>0b010</td>
<td>TRCSSCCR&lt;n&gt;</td>
<td>Single-shot Comparator Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b0:n[2:0]</td>
<td>0b011</td>
<td>TRCSSPCICR&lt;n&gt;</td>
<td>Single-shot Processing Element Comparator Input Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b1:n[2:0]</td>
<td>0b010</td>
<td>TRCSSCSR&lt;n&gt;</td>
<td>Single-shot Comparator Control Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>0b1:n[2:0]</td>
<td>0b00 : 0b000</td>
<td>TRCRSCTRLR&lt;n&gt;</td>
<td>Resource Selection Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0001</td>
<td>n[3:0]</td>
<td>0b00 : 0b000</td>
<td>TRCRSCTRLR&lt;n&gt;</td>
<td>Address Comparator Value Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0010</td>
<td>n[2:0]:0b0</td>
<td>0b00 : 0b000</td>
<td>TRCACVR&lt;n&gt;</td>
<td>Address Comparator Value Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0010</td>
<td>n[2:0]:0b0</td>
<td>0b01 : 0b000</td>
<td>TRCACATR&lt;n&gt;</td>
<td>Address Comparator Access Type Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b010</td>
<td>TRCICIDCTRLR0</td>
<td>Comparator Identification Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0001</td>
<td>0b010</td>
<td>TRCICIDCTRLR1</td>
<td>Comparator Identification Control Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0010</td>
<td>0b010</td>
<td>TRCVMDCTLR0</td>
<td>Virtual Context Identifier Comparator Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>0b0011</td>
<td>0b010</td>
<td>TRCVMDCTLR1</td>
<td>Virtual Context Identifier Comparator Control Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>n[2:0]:0b0</td>
<td>0b000</td>
<td>TRCCIDCVR&lt;n&gt;</td>
<td>Context Identifier Comparator Value Register&lt;n&gt;</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0011</td>
<td>n[2:0]:0b0</td>
<td>0b001</td>
<td>TRCVMDCVR&lt;n&gt;</td>
<td>Virtual Context Identifier Comparator Value Register&lt;n&gt;</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b0010</td>
<td>0b111</td>
<td>TRCDEVID</td>
<td>Device Configuration Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1000</td>
<td>0b110</td>
<td>TRCCLAIMSET</td>
<td>Claim Tag Set Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1001</td>
<td>0b110</td>
<td>TRCCLAIMCLR</td>
<td>Claim Tag Clear Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1110</td>
<td>0b110</td>
<td>TRCAUTHSTATUS</td>
<td>Authentication Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b001</td>
<td>0b0111</td>
<td>0b1111</td>
<td>0b110</td>
<td>TRCDEVARCH</td>
<td>Device Architecture Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
<td>MDCCSR_EL0</td>
<td>Monitor DCC Status Register</td>
</tr>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b000</td>
<td>DBGDTR_EL0</td>
<td>Debug Data Transfer Register, half-duplex</td>
</tr>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b000</td>
<td>DBGDTRRX_EL0</td>
<td>Debug Data Transfer Register, Receive</td>
</tr>
<tr>
<td>0b10</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b000</td>
<td>DBGDTRTX_EL0</td>
<td>Debug Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>0b10</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
<td>DBGVCR32_EL2</td>
<td>Debug Vector Catch Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td>MIDR_EL1</td>
<td>Main ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
<td>MPIDR_EL1</td>
<td>Multiprocessor Affinity Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b110</td>
<td>REVIDR_EL1</td>
<td>Revision Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b000</td>
<td>ID_PFR0_EL1</td>
<td>AArch32 Processor Feature Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-----------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b001</td>
<td>ID_PFR1_EL1</td>
<td>AArch32 Processor Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b010</td>
<td>ID_DFR0_EL1</td>
<td>AArch32 Feature Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b011</td>
<td>ID_AFR0_EL1</td>
<td>AArch32 Auxiliary Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b100</td>
<td>ID_MMFR0_EL1</td>
<td>AArch32 Memory Feature Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b101</td>
<td>ID_MMFR1_EL1</td>
<td>AArch32 Memory Feature Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b110</td>
<td>ID_MMFR2_EL1</td>
<td>AArch32 Memory Feature Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0001</td>
<td>0b111</td>
<td>ID_MMFR3_EL1</td>
<td>AArch32 Memory Feature Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b000</td>
<td>ID_ISAR0_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b001</td>
<td>ID_ISAR1_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b010</td>
<td>ID_ISAR2_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b011</td>
<td>ID_ISAR3_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b100</td>
<td>ID_ISAR4_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b101</td>
<td>ID_ISAR5_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b110</td>
<td>ID_MMFR4_EL1</td>
<td>AArch32 Memory Feature Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0010</td>
<td>0b111</td>
<td>ID_ISAR6_EL1</td>
<td>AArch32 Instruction Set Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b000</td>
<td>MVFR0_EL1</td>
<td>AArch32 and VFP Feature Registers</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>--------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b001</td>
<td>MVFR1_EL1</td>
<td>AArch32 and VFP Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b010</td>
<td>MVFR2_EL1</td>
<td>AArch32 and VFP Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b100</td>
<td>ID_PFR2_EL1</td>
<td>AArch32 Processor Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b101</td>
<td>ID_DFR1_EL1</td>
<td>Debug Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0011</td>
<td>0b110</td>
<td>ID_MMFR5_EL1</td>
<td>AArch32 Memory Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b000</td>
<td>ID_AA64PFR0_EL1</td>
<td>AArch64 Processor Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b001</td>
<td>ID_AA64PFR1_EL1</td>
<td>AArch64 Processor Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0100</td>
<td>0b100</td>
<td>ID_AA64ZFR0_EL1</td>
<td>SVE Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b000</td>
<td>ID_AA64DFR0_EL1</td>
<td>AArch64 Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b001</td>
<td>ID_AA64DFR1_EL1</td>
<td>AArch64 Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b100</td>
<td>ID_AA64AFR0_EL1</td>
<td>AArch64 Auxiliary Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0101</td>
<td>0b101</td>
<td>ID_AA64AFR1_EL1</td>
<td>AArch64 Auxiliary Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b000</td>
<td>ID_AA64ISAR0_EL1</td>
<td>AArch64 Instruction Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0110</td>
<td>0b001</td>
<td>ID_AA64ISAR1_EL1</td>
<td>AArch64 Instruction Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b000</td>
<td>ID_AA64MMFR0_EL1</td>
<td>AArch64 Memory Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b001</td>
<td>ID_AA64MMFR1_EL1</td>
<td>AArch64 Memory Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0000</td>
<td>0b0111</td>
<td>0b010</td>
<td>ID_AA64MMFR2_EL1</td>
<td>AArch64 Memory Feature Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
<td>SCTLR_EL1</td>
<td>System Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
<td>ACTLR_EL1</td>
<td>Auxiliary Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b010</td>
<td>CPACR_EL1</td>
<td>Architectural Feature Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b101</td>
<td>RGSR_EL1</td>
<td>Random Allocation Seed Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b110</td>
<td>GCR_EL1</td>
<td>Tag Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
<td>ZCR_EL1</td>
<td>SVE Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b001</td>
<td>TRFCR_EL1</td>
<td>Trace Filter Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
<td>TTBR0_EL1</td>
<td>Translation Table Base Register 0 (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
<td>TTBR1_EL1</td>
<td>Translation Table Base Register 1 (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
<td>TCR_EL1</td>
<td>Translation Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b000</td>
<td>APIAKeyLo_EL1</td>
<td>Pointer Authentication Key A for Instruction (bits[63:0])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b001</td>
<td>APIAKeyHi_EL1</td>
<td>Pointer Authentication Key A for Instruction (bits[127:64])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
<td>APIBKeyLo_EL1</td>
<td>Pointer Authentication Key B for Instruction (bits[63:0])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b011</td>
<td>APIBKeyHi_EL1</td>
<td>Pointer Authentication Key B for Instruction (bits[127:64])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b000</td>
<td>APDAKeyLo_EL1</td>
<td>Pointer Authentication Key A for Data (bits[63:0])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b001</td>
<td>APDAKeyHi_EL1</td>
<td>Pointer Authentication Key A for Data (bits[127:64])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b00</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b010</td>
<td>APDBKeyLo_EL1</td>
<td>Pointer Authentication Key B for Data (bits[63:0])</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b011</td>
<td>APDBKeyHi_EL1</td>
<td>Pointer Authentication Key B for Data (bits[63:0])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0011</td>
<td>0b000</td>
<td>APGAKeyLo_EL1</td>
<td>Pointer Authentication Key B for Data (bits[127:64])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0010</td>
<td>0b0011</td>
<td>0b001</td>
<td>APGAKeyHi_EL1</td>
<td>Pointer Authentication Key A for Code (bits[63:0])</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
<td>SPSR_EL1</td>
<td>Saved Program Status Register (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
<td>ELR_EL1</td>
<td>Exception Link Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
<td>SP_EL0</td>
<td>Stack Pointer (EL0)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b000</td>
<td>SPSel</td>
<td>Stack Pointer Select</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b010</td>
<td>CurrentEL</td>
<td>Current Exception</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b011</td>
<td>PAN</td>
<td>Privileged Access Never</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b100</td>
<td>UAO</td>
<td>User Access Override</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
<td>ICC_PMR_EL1</td>
<td>Interrupt Controller Interrupt Priority Mask Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
<td>AFSR0_EL1</td>
<td>Auxiliary Status Register 0 (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
<td>AFSR1_EL1</td>
<td>Auxiliary Status Register 1 (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
<td>ESR_EL1</td>
<td>Exception Syndrome Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
<td>ERRIDR_EL1</td>
<td>Error Record Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b001</td>
<td>ERRSELR_EL1</td>
<td>Error Record Select Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b000</td>
<td>ERXFR_EL1</td>
<td>Selected Error Record Feature Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b001</td>
<td>ERXCTLRL_EL1</td>
<td>Selected Error Record Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b010</td>
<td>ERXSTATUS_EL1</td>
<td>Selected Error Record Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b011</td>
<td>ERXADDR_EL1</td>
<td>Selected Error Record Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b100</td>
<td>ERXPFGF_EL1</td>
<td>Selected Pseudo-fault Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>---------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b101</td>
<td>ERXPFGCTL_EL1</td>
<td>Selected Pseudo-fault Generation Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0100</td>
<td>0b110</td>
<td>ERXPFGCDN_EL1</td>
<td>Selected Pseudo-fault Generation Countdown Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b000</td>
<td>ERXMISC0_EL1</td>
<td>Selected Record Miscellaneous Register 0</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b001</td>
<td>ERXMISC1_EL1</td>
<td>Selected Record Miscellaneous Register 1</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b010</td>
<td>ERXMISC2_EL1</td>
<td>Selected Record Miscellaneous Register 2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0101</td>
<td>0b011</td>
<td>ERXMISC3_EL1</td>
<td>Selected Record Miscellaneous Register 3</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
<td>TFSR_EL1</td>
<td>Tag Fail Status Register (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b001</td>
<td>TFSRE0_EL1</td>
<td>Tag Fail Status Register (EL0)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
<td>FAR_EL1</td>
<td>Fault Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b0111</td>
<td>0b0100</td>
<td>0b000</td>
<td>PAR_EL1</td>
<td>Physical Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
<td>PMSCR_EL1</td>
<td>Statistical Profiling Control Register (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b010</td>
<td>PMSICR_EL1</td>
<td>Sampling Interval Counter Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b011</td>
<td>PMSIRR_EL1</td>
<td>Sampling Interval Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b100</td>
<td>PMSFCR_EL1</td>
<td>Sampling Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b101</td>
<td>PMSEVFR_EL1</td>
<td>Sampling Filter Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b110</td>
<td>PMSLATFR_EL1</td>
<td>Sampling Latency Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b111</td>
<td>PMSIDR_EL1</td>
<td>Sampling Profiling Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b000</td>
<td>PMBLIMITR_EL1</td>
<td>Profiling Limit Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b001</td>
<td>PMBPTR_EL1</td>
<td>Profiling Write Pointer Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b011</td>
<td>PMBSR_EL1</td>
<td>Profiling Status/syndrome Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1010</td>
<td>0b111</td>
<td>PMBIDR_EL1</td>
<td>Profiling ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b000</td>
<td>TRBLIMITR_EL1</td>
<td>Trace Buffer Limit Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b001</td>
<td>TRBPTR_EL1</td>
<td>Trace Buffer Write Pointer Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b010</td>
<td>TRBBASER_EL1</td>
<td>Trace Buffer Base Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b011</td>
<td>TRBSR_EL1</td>
<td>Trace Buffer Status/syndrome Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b100</td>
<td>TRBMAR_EL1</td>
<td>Trace Buffer Memory Attribute Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b110</td>
<td>TRBTRG_EL1</td>
<td>Trace Buffer Trigger Counter Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1011</td>
<td>0b111</td>
<td>TRBIDR_EL1</td>
<td>Trace Buffer ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b001</td>
<td>PMINTENSESET_EL1</td>
<td>Performance Monitors Interrupt Enable Set register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b010</td>
<td>PMINTENCLR_EL1</td>
<td>Performance Monitors Interrupt Enable Clear register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1001</td>
<td>0b1110</td>
<td>0b110</td>
<td>PMMIR_EL1</td>
<td>Performance Monitors Machine Identification Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
<td>MAIR_EL1</td>
<td>Memory Attribute Indirection Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
<td>AMAIR_EL1</td>
<td>Auxiliary Memory Attribute Indirection Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b000</td>
<td>LORSA_EL1</td>
<td>LORegion Address</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b001</td>
<td>LOREA_EL1</td>
<td>LORegion Address</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b010</td>
<td>LORN_EL1</td>
<td>LORegion Number</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b011</td>
<td>LORC_EL1</td>
<td>LORegion Control</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b100</td>
<td>MPAMIDR_EL1</td>
<td>MPAM ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b111</td>
<td>LORID_EL1</td>
<td>LORegion (EL1)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
<td>MPAM1_EL1</td>
<td>MPAM1 Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b001</td>
<td>MPAM0_EL1</td>
<td>MPAM0 Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
<td>VBAR_EL1</td>
<td>Vector Base Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
<td>RVBAR_EL1</td>
<td>Reset Vector Base Address Register if EL2 and EL3 not implemented</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
<td>RMR_EL1</td>
<td>Reset Management Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b000</td>
<td>ISR_EL1</td>
<td>Interrupt Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
<td>DISR_EL1</td>
<td>Deferred Interrupt Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b000</td>
<td>ICC_JAR0_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b001</td>
<td>ICC_EOIR0_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b010</td>
<td>ICC_HPPIR0_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b011</td>
<td>ICC_BPR0_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b1:n[1:0]</td>
<td>ICC_AP0&lt;n&gt;_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
<td>ICC_AP1&lt;n&gt;_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
<td>ICC_DIR_EL1</td>
<td>Interrupt Controller</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>--------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
<td>ICC_RPR_EL1</td>
<td>Interrupt Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b101</td>
<td>ICC_SGI1R_EL1</td>
<td>Interrupt Controller Software Generated Interrupt 1 Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b110</td>
<td>ICC_ASGI1R_EL1</td>
<td>Interrupt Controller Software Generated Interrupt 1 Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b111</td>
<td>ICC_SGI0R_EL1</td>
<td>Interrupt Controller Software Generated Interrupt 0 Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b000</td>
<td>ICC_IAR1_EL1</td>
<td>Interrupt Controller Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b001</td>
<td>ICC_EOIR1_EL1</td>
<td>Interrupt Controller Of Interrupt Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b010</td>
<td>ICC_HPPIR1_EL1</td>
<td>Interrupt Controller Highest Pending Interrupt Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b011</td>
<td>ICC_BPR1_EL1</td>
<td>Interrupt Controller Binary P. Pending Interrupt Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
<td>ICC_CTLR_EL1</td>
<td>Interrupt Controller Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
<td>ICC_SRE_EL1</td>
<td>Interrupt Controller System Register Enable Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b110</td>
<td>ICC_IGRPEN0_EL1</td>
<td>Interrupt Controller Interrupt 0 Enable register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
<td>ICC_IGRPEN1_EL1</td>
<td>Interrupt Controller Interrupt 1 Enable register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
<td>CONTEXTIDR_EL1</td>
<td>Context ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b100</td>
<td>TPIDR_EL1</td>
<td>EL1 Software Thread ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
<td>SCXTNUM_EL1</td>
<td>EL1 Read/Write Context Number</td>
</tr>
<tr>
<td>0b11</td>
<td>0b000</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
<td>CNTKCTL_EL1</td>
<td>Counter Kernel Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td>CCSIDR_EL1</td>
<td>Current Size ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b010</td>
<td>CCSIDR2_EL1</td>
<td>Current Size ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b100</td>
<td>GMID_EL1</td>
<td>Multiple transfer register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b001</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
<td>AIDR_EL1</td>
<td>Auxiliary Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b010</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td>CSSELR_EL1</td>
<td>Cache Select Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b001</td>
<td>CTR_EL0</td>
<td>Cache Type Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b111</td>
<td>DCZID_EL0</td>
<td>Data Cache Zero ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0010</td>
<td>0b0100</td>
<td>0b000</td>
<td>RNDR</td>
<td>Random Number</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0010</td>
<td>0b0100</td>
<td>0b001</td>
<td>RNDRRS</td>
<td>Reseeded Random Number</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b000</td>
<td>NZCV</td>
<td>Condition Flags</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b001</td>
<td>DAIF</td>
<td>Interrupt Bits</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b101</td>
<td>DIT</td>
<td>Data Independent Timing</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b110</td>
<td>SSBS</td>
<td>Speculative Store By Safe</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0010</td>
<td>0b111</td>
<td>TCO</td>
<td>Tag Check Override</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0100</td>
<td>0b000</td>
<td>FPCR</td>
<td>Floating-point Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0100</td>
<td>0b001</td>
<td>FPSR</td>
<td>Floating-point Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b000</td>
<td>DSPSR_EL0</td>
<td>Debug Save Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b0100</td>
<td>0b0101</td>
<td>0b001</td>
<td>DLR_EL0</td>
<td>Debug Link register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td>0b000</td>
<td>PMCR_EL0</td>
<td>Performance Monitor Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>op2</td>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMCNTENSET_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMCNTENCLR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMOVSCLR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMOVSCLR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMSCWLC_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMSELR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMCEID0_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1100</td>
<td><strong>PMCEID1_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1101</td>
<td><strong>PMCCNTR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1101</td>
<td><strong>PMXEVTPYPER_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1101</td>
<td><strong>PMXEVNCNTR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1110</td>
<td><strong>PMUSERERNR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1001</td>
<td>0b1110</td>
<td><strong>PMOVSSSET_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td><strong>TPIDR_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td><strong>TPIDRRO_EL0</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
<td>SCXTNUM_EL0</td>
<td>EL0 Read-Write. Software Context Number</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b000</td>
<td>AMCR_EL0</td>
<td>Activity Monitors Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b001</td>
<td>AMCFGR_EL0</td>
<td>Activity Monitors Configure Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b010</td>
<td>AMCGCR_EL0</td>
<td>Activity Monitors Counter Configure Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b011</td>
<td>AMUSERENR_EL0</td>
<td>Activity Monitors Enable Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b100</td>
<td>AMCNTENCLR0_EL0</td>
<td>Activity Monitors Enable Clear Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0010</td>
<td>0b101</td>
<td>AMCNTENSET0_EL0</td>
<td>Activity Monitors Enable Set Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b000</td>
<td>AMCNTENCLR1_EL0</td>
<td>Activity Monitors Enable Clear Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b0011</td>
<td>0b001</td>
<td>AMCNTENSET1_EL0</td>
<td>Activity Monitors Enable Set Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b010:n[3]</td>
<td>n[2:0]</td>
<td>AMEVCNTR0&lt;n&gt;_EL0</td>
<td>Activity Monitors Counter Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b011:n[3]</td>
<td>n[2:0]</td>
<td>AMEVTYPER0&lt;n&gt;_EL0</td>
<td>Activity Monitors Type Register 0</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b110:n[3]</td>
<td>n[2:0]</td>
<td>AMEVCNTR1&lt;n&gt;_EL0</td>
<td>Activity Monitors Counter Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1101</td>
<td>0b111:n[3]</td>
<td>n[2:0]</td>
<td>AMEVTYPER1&lt;n&gt;_EL0</td>
<td>Activity Monitors Type Register 1</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b000</td>
<td>CNTFRO_EL0</td>
<td>Counter Frequency register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b001</td>
<td>CNTPCT_EL0</td>
<td>Counter Physical register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b010</td>
<td>CNTVCT_EL0</td>
<td>Counter-Timer Virtual Count register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b101</td>
<td>CNTPCTSS_EL0</td>
<td>Counter-Self-Synchronized Physical Count register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b110</td>
<td>CNTVCTSS_EL0</td>
<td>Counter-Self-Synchronized Virtual Count register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
<td>CNTP_TVAL_EL0</td>
<td>Counter-Physical Timer Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
<td>CNTP_CTL_EL0</td>
<td>Counter-Physical Compare register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
<td>CNTP_CVAL_EL0</td>
<td>Counter-Physical Compare Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
<td>CNTV_TVAL_EL0</td>
<td>Counter-Virtual Timer Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
<td>CNTV_CTL_EL0</td>
<td>Counter-Virtual Timer Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
<td>CNTV_CVAL_EL0</td>
<td>Counter-Virtual Timer Compare Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b10:n[4:3] n[2:0]</td>
<td>PMEVCNTR&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Counter Register</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b1111</td>
<td>0b111</td>
<td>PMCCFILTR_EL0</td>
<td>Performance Monitors Cycle Count Filter Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b011</td>
<td>0b1110</td>
<td>0b11:n[4:3] n[2:0]</td>
<td>PMEVTYPER&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Type Register</td>
<td></td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b000</td>
<td>VPIDR_EL2</td>
<td>Virtualization Processor ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0000</td>
<td>0b0000</td>
<td>0b101</td>
<td>VMPIDR_EL2</td>
<td>Virtualization Multiprocessor ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
<td>SCTLR_EL2</td>
<td>System Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
<td>ACTLR_EL2</td>
<td>Auxiliary Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
<td>HCR_EL2</td>
<td>Hypervisor Configuration Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
<td>MDCR_EL2</td>
<td>Monitor Configuration Register</td>
</tr>
</tbody>
</table>
### Register selectors

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
<td>CPTR_EL2</td>
<td>Architectural Feature Trap Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b011</td>
<td>HSTR_EL2</td>
<td>Hypervisor System Trap Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b100</td>
<td>HFGTR_EL2</td>
<td>Hypervisor Fine-Grained Read Trap Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b101</td>
<td>HFGWTR_EL2</td>
<td>Hypervisor Fine-Grained Write Trap Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b110</td>
<td>HFGITR_EL2</td>
<td>Hypervisor Fine-Grained Instruction Trap Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b111</td>
<td>HACR_EL2</td>
<td>Hypervisor Auxiliary Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
<td>ZCR_EL2</td>
<td>SVE Control Register EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
<td>TRFCR_EL2</td>
<td>Trace Filter Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0011</td>
<td>0b001</td>
<td>SDER32_EL2</td>
<td>AArch32 Secure Debug Enable Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
<td>TTBR0_EL2</td>
<td>Translation Table Base Register (EL2)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b001</td>
<td>TTBR1_EL2</td>
<td>Translation Table Base Register (EL2)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
<td>TCR_EL2</td>
<td>Translation Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b000</td>
<td>VTTBR_EL2</td>
<td>Virtualization Translation Table Base Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0001</td>
<td>0b010</td>
<td>VTCR_EL2</td>
<td>Virtualization Translation Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0010</td>
<td>0b000</td>
<td>VNCR_EL2</td>
<td>Virtualization Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0110</td>
<td>0b000</td>
<td>VSTTBR_EL2</td>
<td>Virtualization Secure Translations Table Base Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0010</td>
<td>0b0110</td>
<td>0b010</td>
<td>VSTCR_EL2</td>
<td>Virtualization Secure Translation System Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b000</td>
<td>DACR32_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b001</td>
<td>0b100</td>
<td>HDFGRTR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b001</td>
<td>0b101</td>
<td>HDFGWTR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0011</td>
<td>0b0000</td>
<td>0b010</td>
<td>0b110</td>
<td>HAFGRTR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b000</td>
<td>SPSR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
<td>0b001</td>
<td>ELR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b000</td>
<td>SP_EL1</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b000</td>
<td>SPSR_irq</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b000</td>
<td>SPSR_abt</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b010</td>
<td>0b010</td>
<td>SPSR_und</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0100</td>
<td>0b0011</td>
<td>0b011</td>
<td>0b011</td>
<td>SPSR_fiq</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b001</td>
<td>0b001</td>
<td>IFSR32_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b000</td>
<td>AFSR0_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b000</td>
<td>AFSR1_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
<td>0b000</td>
<td>ESR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b011</td>
<td>0b011</td>
<td>VSESRL_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b000</td>
<td>FPXEC32_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
<td>0b000</td>
<td>TFSR_EL2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
<td>0b000</td>
<td>FAR_EL2</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>---------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b100</td>
<td>HPFAR_EL2</td>
<td>Hypervisor Fault Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1001</td>
<td>0b1001</td>
<td>0b000</td>
<td>PMSCR_EL2</td>
<td>Statistical Profiling Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
<td>MAIR_EL2</td>
<td>Memory Attribute Indirection Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
<td>AMAIR_EL2</td>
<td>Auxiliary Memory Attribute Indirection Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b000</td>
<td>MPAMHCR_EL2</td>
<td>MPAM Hypervisor Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0100</td>
<td>0b001</td>
<td>MPAMVPMV_EL2</td>
<td>MPAM Virtual Partition Mapping Valid Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
<td>MPAM2_EL2</td>
<td>MPAM2 Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b000</td>
<td>MPAMVPM0_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 0</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b001</td>
<td>MPAMVPM1_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 1</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b010</td>
<td>MPAMVPM2_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 2</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b011</td>
<td>MPAMVPM3_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 3</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b100</td>
<td>MPAMVPM4_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 4</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b101</td>
<td>MPAMVPM5_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 5</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b110</td>
<td>MPAMVPM6_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 6</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1010</td>
<td>0b0110</td>
<td>0b111</td>
<td>MPAMVPM7_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 7</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
<td>VBAR_EL2</td>
<td>Vector Base Address Register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-----------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
<td>RVBAR_EL2</td>
<td>Reset Vector Base Address Register (if EL3 not implemented)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
<td>RMR_EL2</td>
<td>Reset Management Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b0001</td>
<td>0b001</td>
<td>VDISR_EL2</td>
<td>Virtual Deferred Interrupt Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1000</td>
<td>0b0:n[1:0]</td>
<td>ICH_AP0R&lt;n&gt;_EL2</td>
<td>Interrupt Controller Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b0:n[1:0]</td>
<td>ICH_AP1R&lt;n&gt;_EL2</td>
<td>Interrupt Controller Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1001</td>
<td>0b101</td>
<td>ICC_SRE_EL2</td>
<td>Interrupt Controller System Register Enable register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b000</td>
<td>ICH_HCR_EL2</td>
<td>Interrupt Controller Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b001</td>
<td>ICH_VTR_EL2</td>
<td>Interrupt Controller Type Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b010</td>
<td>ICH_MISR_EL2</td>
<td>Interrupt Controller Maintenance Interrupt Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b011</td>
<td>ICH_EISR_EL2</td>
<td>Interrupt Controller End of Interrupt Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b101</td>
<td>ICH_ELRSR_EL2</td>
<td>Interrupt Controller Empty List Register Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1011</td>
<td>0b111</td>
<td>ICH_VMCR_EL2</td>
<td>Interrupt Controller Virtual Machine Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1100</td>
<td>0b1010</td>
<td>n[2:0]</td>
<td>ICH_LR&lt;n&gt;_EL2</td>
<td>Interrupt Controller Registers</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b001</td>
<td>CONTEXTIDR_EL2</td>
<td>Context Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
<td>TPIDR_EL2</td>
<td>EL2 Software Thread ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
<td>SCXTNUM_EL2</td>
<td>EL2 Read Software</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b100:n[3]</td>
<td>n[2:0]</td>
<td>AMEVNTVOFF0&lt;n&gt;_EL2</td>
<td>Activity Monitors Counter Offset Register 0</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1101</td>
<td>0b101:n[3]</td>
<td>n[2:0]</td>
<td>AMEVNTVOFF1&lt;n&gt;_EL2</td>
<td>Activity Monitors Counter Offset Register 1</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b011</td>
<td>CNTVOFF_EL2</td>
<td>Counter Virtual Offset register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0000</td>
<td>0b110</td>
<td>CNTPOFF_EL2</td>
<td>Counter Physical Offset register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0001</td>
<td>0b000</td>
<td>CNTHCTL_EL2</td>
<td>Counter Hypervisor Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
<td>CNTVP_TVAL_EL2</td>
<td>Counter Virtual Timer Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
<td>CNTHP_CTL_EL2</td>
<td>Counter Hypervisor Physical Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
<td>CNTHP_CVAL_EL2</td>
<td>Counter Physical Compare register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b000</td>
<td>CNTHV_TVAL_EL2</td>
<td>Counter Virtual Timer Value register (EL2)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b001</td>
<td>CNTHV_CTL_EL2</td>
<td>Counter Virtual Timer Control register (EL2)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0011</td>
<td>0b010</td>
<td>CNTHV_CVAL_EL2</td>
<td>Counter Virtual Timer Compare register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0100</td>
<td>0b000</td>
<td>CNTHVS_TVAL_EL2</td>
<td>Counter Secure Virtual Timer Timer Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0100</td>
<td>0b001</td>
<td>CNTHVS_CTL_EL2</td>
<td>Counter Secure Virtual Timer Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0100</td>
<td>0b010</td>
<td>CNTHVS_CVAL_EL2</td>
<td>Counter Secure Virtual Timer Compare register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0101</td>
<td>0b000</td>
<td>CNTHPS_TVAL_EL2</td>
<td>Counter Secure Physical Timer Value register</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>Register selectors</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Name</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>--------------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0101</td>
<td>0b001</td>
<td>CNTHPS_CTL_EL2</td>
<td>Timer Value register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b100</td>
<td>0b1110</td>
<td>0b0101</td>
<td>0b010</td>
<td>CNTHPS_CVAL_EL2</td>
<td>Counter Secure Physical Timer Compare register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b000</td>
<td>SCTRLR_EL3</td>
<td>System Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0000</td>
<td>0b001</td>
<td>ACTLR_EL3</td>
<td>Auxiliary Control register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b000</td>
<td>SCR_EL3</td>
<td>Secure Configuration register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b001</td>
<td>SDER32_EL3</td>
<td>AArch32 Secure I/F Enable Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0001</td>
<td>0b010</td>
<td>CPTR_EL3</td>
<td>Architectural Feature Trap register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0010</td>
<td>0b000</td>
<td>ZCR_EL3</td>
<td>SVE Control Register EL3</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0001</td>
<td>0b0011</td>
<td>0b001</td>
<td>MDCR_EL3</td>
<td>Monitor Debug Configuration register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b000</td>
<td>TTBR0_EL3</td>
<td>Translation Table Base Register 0 (EL3)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0010</td>
<td>0b0000</td>
<td>0b010</td>
<td>TCR_EL3</td>
<td>Translation Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b000</td>
<td>SPSR_EL3</td>
<td>Saved Processor Status Register (EL3)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0000</td>
<td>0b001</td>
<td>ELR_EL3</td>
<td>Exception Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0100</td>
<td>0b0001</td>
<td>0b000</td>
<td>SP_EL2</td>
<td>Stack Pointer (EL2)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b000</td>
<td>AFSR0_EL3</td>
<td>Auxiliary Status Register 0 (EL3)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0001</td>
<td>0b001</td>
<td>AFSR1_EL3</td>
<td>Auxiliary Status Register 1 (EL3)</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0010</td>
<td>0b000</td>
<td>ESR_EL3</td>
<td>Exception Syndrome Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0101</td>
<td>0b0110</td>
<td>0b000</td>
<td>TFSR_EL3</td>
<td>Tag Fail Status Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b0110</td>
<td>0b0000</td>
<td>0b000</td>
<td>FAR_EL3</td>
<td>Fault Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0010</td>
<td>0b000</td>
<td>MAIR_EL3</td>
<td>Memory Attribute Register</td>
</tr>
</tbody>
</table>
### Register selectors

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0011</td>
<td>0b000</td>
<td>AMAIR_EL3</td>
<td>Indirect Memory Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1010</td>
<td>0b0101</td>
<td>0b000</td>
<td>MPAM3_EL3</td>
<td>Auxiliary Memory Attribute Indirect Memory Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b000</td>
<td>VBAR_EL3</td>
<td>Vector Base Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b001</td>
<td>RVBAR_EL3</td>
<td>Reset Vector Base Address Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b0000</td>
<td>0b010</td>
<td>RMR_EL3</td>
<td>Reset Management Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b100</td>
<td>ICC_CTLR_EL3</td>
<td>Interrupt Controller Control Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b101</td>
<td>ICC_SRE_EL3</td>
<td>Interrupt Controller System Register Enable register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1100</td>
<td>0b1100</td>
<td>0b111</td>
<td>ICC_IGRPEN1_EL3</td>
<td>Interrupt Controller Interrupt Group 1 Enable register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b010</td>
<td>TPIDR_EL3</td>
<td>EL3 Software Thread ID Register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b110</td>
<td>0b1101</td>
<td>0b0000</td>
<td>0b111</td>
<td>SCXTNUM_EL3</td>
<td>EL3 Read/Write Software Context Number</td>
</tr>
<tr>
<td>0b11</td>
<td>0b111</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b000</td>
<td>CNTPS_TVAL_EL1</td>
<td>Counter-Physical Secure Timer TimerValue register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b111</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b001</td>
<td>CNTPS_CTL_EL1</td>
<td>Counter-Physical Secure Timer Compare register</td>
</tr>
<tr>
<td>0b11</td>
<td>0b111</td>
<td>0b1110</td>
<td>0b0010</td>
<td>0b010</td>
<td>CNTPS_CVAL_EL1</td>
<td>Counter-Physical Secure Timer Compare register</td>
</tr>
</tbody>
</table>

### Accessed using TLBI:

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b1111</td>
<td>TLBI_VMALLE1OS</td>
<td>TLB Invalidate by VMID, All at</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Rt</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>----</td>
<td>---------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE1OS</td>
<td>TLB Invalidate by VA, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b010</td>
<td>-</td>
<td>TLBI ASIDE1OS</td>
<td>TLB Invalidate by ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b011</td>
<td>-</td>
<td>TLBI VAAE1OS</td>
<td>TLB Invalidate by VA, All ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VAE1OS</td>
<td>TLB Invalidate by VA, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b111</td>
<td>-</td>
<td>TLBI VAALE1OS</td>
<td>TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE1IS</td>
<td>TLB Range Invalidate by VA, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b011</td>
<td>-</td>
<td>TLBI RVAAE1IS</td>
<td>TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE1IS</td>
<td>TLB Range Invalidate by VA, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b111</td>
<td>-</td>
<td>TLBI RVAALE1IS</td>
<td>TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b1111</td>
<td>TLBI VMALLE1IS</td>
<td>TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE1IS</td>
<td>TLB Invalidate by VA, EL1, Inner Shareable</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Rt</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b010</td>
<td>-</td>
<td>TLBI ASIDE1IS</td>
<td>TLB Invalidate by ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b011</td>
<td>-</td>
<td>TLBI VAAE1IS</td>
<td>TLB Invalidate by VA, All ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE1IS</td>
<td>TLB Invalidate by VA, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b111</td>
<td>-</td>
<td>TLBI VAALE1IS</td>
<td>TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE1OS</td>
<td>TLB Range Invalidate by VA, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b011</td>
<td>-</td>
<td>TLBI RVAAE1OS</td>
<td>TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE1OS</td>
<td>TLB Range Invalidate by VA, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b111</td>
<td>-</td>
<td>TLBI RVALE1OS</td>
<td>TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE1</td>
<td>TLB Range Invalidate by VA, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b011</td>
<td>-</td>
<td>TLBI RVAE1</td>
<td>TLB Range Invalidate by VA, All ASID, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE1</td>
<td>TLB Range Invalidate by VA, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b111</td>
<td>-</td>
<td>TLBI RVALE1</td>
<td>TLB Range Invalidate by VA, All ASID, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>0b1111</td>
<td>TLBI VMALLE1</td>
<td>TLB Invalidate by VMID, All at stage 1, EL1</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Rt</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>---------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE1</td>
<td>TLB Invalidate by VA, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b010</td>
<td>-</td>
<td>TLBI ASIDE1</td>
<td>TLB Invalidate by ASID, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b011</td>
<td>-</td>
<td>TLBI VAAE1</td>
<td>TLB Invalidate by VA, All ASID, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE1</td>
<td>TLB Invalidate by VA, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b000</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b111</td>
<td>-</td>
<td>TLBI VAALE1</td>
<td>TLB Invalidate by VA, All ASID, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b001</td>
<td>-</td>
<td>TLBI IPAS2E1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b010</td>
<td>-</td>
<td>TLBI RIPAS2E1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b101</td>
<td>-</td>
<td>TLBI IPAS2LE1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0000</td>
<td>0b110</td>
<td>-</td>
<td>TLBI RIPAS2LE1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b1111</td>
<td>TLBI ALLE2OS</td>
<td>TLB Invalidate All, EL2, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE2OS</td>
<td>TLB Invalidate by VA, EL2, Outer Shareable</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Rt</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>------</td>
<td>-----</td>
<td>------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b100</td>
<td>0b11111</td>
<td>TLBI ALLE1OS</td>
<td>TLB Invalidate All, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE2OS</td>
<td>TLB Invalidate by VA, Last level, EL2, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b110</td>
<td>0b11111</td>
<td>TLBI VMALLS12E1OS</td>
<td>TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE2IS</td>
<td>TLB Range Invalidate by VA, EL2, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE2IS</td>
<td>TLB Range Invalidate by VA, Last level, EL2, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b11111</td>
<td>TLBI ALLE2IS</td>
<td>TLB Invalidate All, EL2, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE2IS</td>
<td>TLB Invalidate by VA, EL2, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b100</td>
<td>0b11111</td>
<td>TLBI ALLE1IS</td>
<td>TLB Invalidate All, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE2IS</td>
<td>TLB Invalidate by VA, Last level, EL2, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b110</td>
<td>0b11111</td>
<td>TLBI VMALLS12E1IS</td>
<td>TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>-</td>
<td>TLBI IPAS2E1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2,</td>
</tr>
</tbody>
</table>
### System Register index by instruction and encoding

<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>CRn</th>
<th>CRm</th>
<th>op2</th>
<th>Rt</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b001</td>
<td>0b010</td>
<td>TLBI IPAS2E1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b010</td>
<td>0b011</td>
<td>TLBI RIPAS2E11OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b011</td>
<td>0b010</td>
<td>TLBI RIPAS2E11OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b100</td>
<td>0b100</td>
<td>TLBI IPAS2LE1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b101</td>
<td>0b101</td>
<td>TLBI IPAS2LE1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b110</td>
<td>0b110</td>
<td>TLBI RIPAS2LE11OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1000</td>
<td>0b111</td>
<td>0b111</td>
<td>TLBI RIPAS2LE11OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1010</td>
<td>0b001</td>
<td>0b010</td>
<td>TLBI RVAE2OS</td>
<td>TLB Range Invalidate by VA, EL2, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b1001</td>
<td>0b101</td>
<td>0b101</td>
<td>TLBI RVALE2OS</td>
<td>TLB Range Invalidate by VA, Last level, EL2, Outer Shareable</td>
</tr>
<tr>
<td>op0</td>
<td>op1</td>
<td>CRn</td>
<td>CRm</td>
<td>op2</td>
<td>Rt</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-------------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE2</td>
<td>TLB Range Invalidate by VA, EL2</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE2</td>
<td>TLB Range Invalidate by VA, Last level, EL2</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>0b11111</td>
<td>TLBI ALLE2</td>
<td>TLB Invalidate All, EL2</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE2</td>
<td>TLB Invalidate by VA, EL2</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b100</td>
<td>0b11111</td>
<td>TLBI ALLE1</td>
<td>TLB Invalidate All, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE2</td>
<td>TLB Invalidate by VA, Last level, EL2</td>
</tr>
<tr>
<td>0b01</td>
<td>0b100</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b110</td>
<td>0b11111</td>
<td>TLBI VMALLS12E1</td>
<td>TLB Invalidate by VMID, All at Stage 1 and 2, EL1</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b000</td>
<td>0b11111</td>
<td>TLBI ALLE3OS</td>
<td>TLB Invalidate All, EL3, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE3OS</td>
<td>TLB Invalidate by VA, EL3, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0001</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE3OS</td>
<td>TLB Invalidate by VA, Last level, EL3, Outer Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE3IS</td>
<td>TLB Range Invalidate by VA, EL3, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0010</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE3IS</td>
<td>TLB Range Invalidate by VA, Last level, EL3, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b000</td>
<td>0b11111</td>
<td>TLBI ALLE3IS</td>
<td>TLB Invalidate All, EL3, Inner Shareable</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE3IS</td>
<td>TLB Invalidate by VA, EL3,</td>
</tr>
</tbody>
</table>

System Register index by instruction and encoding
<table>
<thead>
<tr>
<th>op0</th>
<th>op1</th>
<th>Register selectors</th>
<th>op2</th>
<th>Rt</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0011</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE3IS</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE3OS</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0101</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE3OS</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b001</td>
<td>-</td>
<td>TLBI RVAE3</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0110</td>
<td>0b101</td>
<td>-</td>
<td>TLBI RVALE3</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b000</td>
<td>0b1111</td>
<td>TLBI ALE3</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b001</td>
<td>-</td>
<td>TLBI VAE3</td>
</tr>
<tr>
<td>0b01</td>
<td>0b110</td>
<td>0b1000</td>
<td>0b0111</td>
<td>0b101</td>
<td>-</td>
<td>TLBI VALE3</td>
</tr>
</tbody>
</table>
System Register index by functional group

Below are indexes for registers with the following main functional groups:

- **ID**
- **Memory**
- **Other**
- **Exception**
- **Special**
- **PSTATE**
- **Cache**
- **Address**
- **TLB**
- **PMU**
- **Reset**
- **Thread**
- **IMP DEF**
- **Timer**
- **Debug**
- **CTI**
- **Virt**
- **Secure**
- **Float**
- **Legacy**
- **Trace**
- **GIC**
- **GICD**
- **GICC**
- **GICR**
- **GICH**
- **GTS**
- **RAS**
- **MPAM**
- **Pointer authentication**
- **GIC ITS registers**

In the **ID** functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>CCSIDR</td>
<td>Current Cache Size ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CCSIDR2</td>
<td>Current Cache Size ID Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>CLIDR</td>
<td>Cache Level ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CSSELR</td>
<td>Cache Size Selection Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CTR</td>
<td>Cache Type Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_AFR0</td>
<td>Auxiliary Feature Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_DFR0</td>
<td>Debug Feature Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_DFR1</td>
<td>Debug Feature Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR0</td>
<td>Instruction Set Attribute Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR1</td>
<td>Instruction Set Attribute Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR2</td>
<td>Instruction Set Attribute Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR3</td>
<td>Instruction Set Attribute Register 3</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR4</td>
<td>Instruction Set Attribute Register 4</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR5</td>
<td>Instruction Set Attribute Register 5</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_ISAR6</td>
<td>Instruction Set Attribute Register 6</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_MMFR0</td>
<td>Memory Model Feature Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_MMFR1</td>
<td>Memory Model Feature Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_MMFR2</td>
<td>Memory Model Feature Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_MMFR3</td>
<td>Memory Model Feature Register 3</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_MMFR4</td>
<td>Memory Model Feature Register 4</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_MMFR5</td>
<td>Memory Model Feature Register 5</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_PFR0</td>
<td>Processor Feature Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_PFR1</td>
<td>Processor Feature Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ID_PFR2</td>
<td>Processor Feature Register 2</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>------------------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>AArch32</td>
<td>MIDR</td>
<td>Main ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>MPIDR</td>
<td>Multiprocessor Affinity Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>REVIDR</td>
<td>Revision ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>TCMTR</td>
<td>TCM Type Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBTR</td>
<td>TLB Type Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CCSIDR2_EL1</td>
<td>Current Cache Size ID Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>CCSIDR_EL1</td>
<td>Current Cache Size Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CLIDR_EL1</td>
<td>Cache Level ID Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CSSELK_EL1</td>
<td>Cache Size Selection Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CTR_EL0</td>
<td>Cache Type Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DCZID_EL0</td>
<td>Data Cache Zero ID register</td>
</tr>
<tr>
<td>AArch64</td>
<td>GMID_EL1</td>
<td>Multiple tag transfer ID register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64AFR0_EL1</td>
<td>AArch64 Auxiliary Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64AFR1_EL1</td>
<td>AArch64 Auxiliary Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64DFR0_EL1</td>
<td>AArch64 Debug Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64DFR1_EL1</td>
<td>AArch64 Debug Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64ISAR0_EL1</td>
<td>AArch64 Instruction Set Attribute Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64ISAR1_EL1</td>
<td>AArch64 Instruction Set Attribute Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64MMFR0_EL1</td>
<td>AArch64 Memory Model Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64MMFR1_EL1</td>
<td>AArch64 Memory Model Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64MMFR2_EL1</td>
<td>AArch64 Memory Model Feature Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64PF0_EL1</td>
<td>AArch64 Processor Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64PF1_EL1</td>
<td>AArch64 Processor Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AA64ZF0_EL1</td>
<td>SVE Feature ID register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID AFR0_EL1</td>
<td>AArch32 Auxiliary Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID DFR0_EL1</td>
<td>AArch32 Debug Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID DFR1_EL1</td>
<td>Debug Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR0_EL1</td>
<td>AArch32 Instruction Set Attribute Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR1_EL1</td>
<td>AArch32 Instruction Set Attribute Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR2_EL1</td>
<td>AArch32 Instruction Set Attribute Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR3_EL1</td>
<td>AArch32 Instruction Set Attribute Register 3</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR4_EL1</td>
<td>AArch32 Instruction Set Attribute Register 4</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR5_EL1</td>
<td>AArch32 Instruction Set Attribute Register 5</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID ISAR6_EL1</td>
<td>AArch32 Instruction Set Attribute Register 6</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID MMFR0_EL1</td>
<td>AArch32 Memory Model Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID MMFR1_EL1</td>
<td>AArch32 Memory Model Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID MMFR2_EL1</td>
<td>AArch32 Memory Model Feature Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID MMFR3_EL1</td>
<td>AArch32 Memory Model Feature Register 3</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID MMFR4_EL1</td>
<td>AArch32 Memory Model Feature Register 4</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID MMFR5_EL1</td>
<td>AArch32 Memory Model Feature Register 5</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID PFR0_EL1</td>
<td>AArch32 Processor Feature Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID PFR1_EL1</td>
<td>AArch32 Processor Feature Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ID PFR2_EL1</td>
<td>AArch32 Processor Feature Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>MIDR_EL1</td>
<td>Main ID Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPIDR_EL1</td>
<td>Multiprocessor Affinity Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>REVIDR_EL1</td>
<td>Revision ID Register</td>
</tr>
<tr>
<td>External</td>
<td>EDA32PFR</td>
<td>External Debug AArch32 Processor Feature Register</td>
</tr>
<tr>
<td>External</td>
<td>EDDFR</td>
<td>External Debug Feature Register</td>
</tr>
<tr>
<td>External</td>
<td>EDPR</td>
<td>External Debug Processor Feature Register</td>
</tr>
<tr>
<td>External</td>
<td>MIDR_EL1</td>
<td>Main ID Register</td>
</tr>
</tbody>
</table>

In the Memory functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>AMAIR0</td>
<td>Auxiliary Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>AMAIR1</td>
<td>Auxiliary Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>CONTEXTIDR</td>
<td>Context ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DACR</td>
<td>Domain Access Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HAMAIR0</td>
<td>Hyp Auxiliary Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>HAMAIR1</td>
<td>Hyp Auxiliary Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>HTCR</td>
<td>Hyp Translation Control Register</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>AArch32</td>
<td>HTTBR</td>
<td>Hyp Translation Table Base Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>MAIR0</td>
<td>Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>MAIR1</td>
<td>Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>NMRR</td>
<td>Normal Memory Remap Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PRRR</td>
<td>Primary Region Remap Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>TTBCR</td>
<td>Translation Table Base Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>TTBCR2</td>
<td>Translation Table Base Control Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>TTBR0</td>
<td>Translation Table Base Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>TTBR1</td>
<td>Translation Table Base Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>VTCR</td>
<td>Virtualization Translation Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VTBR</td>
<td>Virtualization Translation Table Base Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>AMAIR_EL1</td>
<td>Auxiliary Memory Attribute Indirection Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AMAIR_EL2</td>
<td>Auxiliary Memory Attribute Indirection Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AMAIR_EL3</td>
<td>Auxiliary Memory Attribute Indirection Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CONTEXTIDR_EL1</td>
<td>Context ID Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CONTEXTIDR_EL2</td>
<td>Context ID Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>DACR32_EL2</td>
<td>Domain Access Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>LORC_EL1</td>
<td>LORegion Control (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>LORC_EL1</td>
<td>LORegion Control (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>LORC_EL3</td>
<td>LORegion Control (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>LOREA_EL1</td>
<td>LORegion End Address (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>LOREA_EL1</td>
<td>LORegion End Address (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>LORC_EL1</td>
<td>LORegion Start Address (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>LORC_EL1</td>
<td>LORegion Start Address (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MAIR_EL1</td>
<td>Memory Attribute Indirection Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MAIR_EL2</td>
<td>Memory Attribute Indirection Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MAIR_EL3</td>
<td>Memory Attribute Indirection Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CONTEXTIDR_EL1</td>
<td>Context ID Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CONTEXTIDR_EL2</td>
<td>Context ID Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TCR_EL1</td>
<td>Translation Control Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TCR_EL2</td>
<td>Translation Control Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TCR_EL3</td>
<td>Translation Control Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBRO_EL1</td>
<td>Translation Table Base Register 0 (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBRO_EL2</td>
<td>Translation Table Base Register 0 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBRO_EL3</td>
<td>Translation Table Base Register 0 (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBR0_EL1</td>
<td>Translation Table Base Register 0 (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBR0_EL2</td>
<td>Translation Table Base Register 0 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBR0_EL3</td>
<td>Translation Table Base Register 0 (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBR1_EL1</td>
<td>Translation Table Base Register 1 (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBR1_EL2</td>
<td>Translation Table Base Register 1 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TTBR1_EL3</td>
<td>Translation Table Base Register 1 (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>VTCR_EL2</td>
<td>Virtualization Translation Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>VTTBR_EL2</td>
<td>Virtualization Translation Table Base Register</td>
</tr>
</tbody>
</table>

In the Other functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>CPACR</td>
<td>Architectural Feature Access Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>SCTLR</td>
<td>System Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CPACR_EL1</td>
<td>Architectural Feature Access Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>SCTLR_EL1</td>
<td>System Control Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SCTLR_EL3</td>
<td>System Control Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ZCR_EL1</td>
<td>SVE Control Register for EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ZCR_EL2</td>
<td>SVE Control Register for EL2</td>
</tr>
<tr>
<td>AArch64</td>
<td>ZCR_EL3</td>
<td>SVE Control Register for EL3</td>
</tr>
</tbody>
</table>

In the Exception functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>ADEFSR</td>
<td>Auxiliary Data Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>AIEFSR</td>
<td>Auxiliary Instruction Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DFAR</td>
<td>Data Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DFRS</td>
<td>Data Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HADEFSR</td>
<td>Hyp Auxiliary Data Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HAIERF</td>
<td>Hyp Auxiliary Instruction Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HDFAR</td>
<td>Hyp Data Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HIFAR</td>
<td>Hyp Instruction Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HIPFAR</td>
<td>Hyp IPA Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HSR</td>
<td>Hyp Syndrome Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HVBAR</td>
<td>Hyp Vector Base Address Register</td>
</tr>
</tbody>
</table>
### System Register index by functional group

#### Exec state Name Description

<table>
<thead>
<tr>
<th>AArch32</th>
<th>IFAR</th>
<th>Instruction Fault Address Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>IFSR</td>
<td>Instruction Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ISR</td>
<td>Interrupt Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>MVBAR</td>
<td>Monitor Vector Base Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VBAR</td>
<td>Vector Base Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR0_EL1</td>
<td>Auxiliary Fault Status Register 0 (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR0_EL2</td>
<td>Auxiliary Fault Status Register 0 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR0_EL3</td>
<td>Auxiliary Fault Status Register 0 (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR1_EL1</td>
<td>Auxiliary Fault Status Register 1 (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR1_EL2</td>
<td>Auxiliary Fault Status Register 1 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR1_EL3</td>
<td>Auxiliary Fault Status Register 1 (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ESR_EL1</td>
<td>Exception Syndrome Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ESR_EL2</td>
<td>Exception Syndrome Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ESR_EL3</td>
<td>Exception Syndrome Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>FAR_EL1</td>
<td>Fault Address Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>FAR_EL2</td>
<td>Fault Address Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>FAR_EL3</td>
<td>Fault Address Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>HPFAR_EL2</td>
<td>Hypervisor IPA Fault Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>IFSR32_EL2</td>
<td>Instruction Fault Status Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ISR_EL1</td>
<td>Interrupt Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>VBAR_EL1</td>
<td>Vector Base Address Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>VBAR_EL2</td>
<td>Vector Base Address Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>VBAR_EL3</td>
<td>Vector Base Address Register (EL3)</td>
</tr>
</tbody>
</table>

#### In the Special functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>DLR</td>
<td>Debug Link Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DSPSR</td>
<td>Debug Saved Program Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ELR_hyp</td>
<td>Exception Link Register (Hyp mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR</td>
<td>Saved Program Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_abt</td>
<td>Saved Program Status Register (Abort mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_fiq</td>
<td>Saved Program Status Register (FIQ mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_hyp</td>
<td>Saved Program Status Register (Hyp mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_irq</td>
<td>Saved Program Status Register (IRQ mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_mon</td>
<td>Saved Program Status Register (Monitor mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_svc</td>
<td>Saved Program Status Register (Supervisor mode)</td>
</tr>
<tr>
<td>AArch32</td>
<td>SPSR_und</td>
<td>Saved Program Status Register (Undefined mode)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ELR_EL1</td>
<td>Exception Link Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ELR_EL2</td>
<td>Exception Link Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ELR_EL3</td>
<td>Exception Link Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_EL1</td>
<td>Saved Program Status Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_EL2</td>
<td>Saved Program Status Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_EL3</td>
<td>Saved Program Status Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_abt</td>
<td>Saved Program Status Register (Abort mode)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_fiq</td>
<td>Saved Program Status Register (FIQ mode)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_irq</td>
<td>Saved Program Status Register (IRQ mode)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSR_und</td>
<td>Saved Program Status Register (Undefined mode)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SP_EL0</td>
<td>Stack Pointer (EL0)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SP_EL1</td>
<td>Stack Pointer (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SP_EL2</td>
<td>Stack Pointer (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SP_EL3</td>
<td>Stack Pointer (EL3)</td>
</tr>
</tbody>
</table>

#### In the PSTATE functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>APSR</td>
<td>Application Program Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CPSR</td>
<td>Current Program Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CurrentEL</td>
<td>Current Exception Level</td>
</tr>
<tr>
<td>AArch64</td>
<td>DAIF</td>
<td>Interrupt Mask Bits</td>
</tr>
<tr>
<td>AArch64</td>
<td>NZCV</td>
<td>Condition Flags</td>
</tr>
</tbody>
</table>
### AArch64

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>PAN</td>
<td>Privileged Access Never</td>
</tr>
<tr>
<td>AArch64</td>
<td>SPSel</td>
<td>Stack Pointer Select</td>
</tr>
<tr>
<td>AArch64</td>
<td>SSB5</td>
<td>Speculative Store Bypass Safe</td>
</tr>
<tr>
<td>AArch64</td>
<td>TCO</td>
<td>Tag Check Override</td>
</tr>
<tr>
<td>AArch64</td>
<td>UAQ</td>
<td>User Access Override</td>
</tr>
</tbody>
</table>

**In the Cache functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>BPIALL</td>
<td>Branch Predictor Invalidate All</td>
</tr>
<tr>
<td>AArch32</td>
<td>BPIALLIS</td>
<td>Branch Predictor Invalidate All, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>BPIMV32</td>
<td>Branch Predictor Invalidate by VA</td>
</tr>
<tr>
<td>AArch32</td>
<td>DCIMMVAC4</td>
<td>Data Cache line Clean and Invalidate by VA to PoC</td>
</tr>
<tr>
<td>AArch32</td>
<td>DCIMISW</td>
<td>Data Cache line Clean and Invalidate by Set/Way</td>
</tr>
<tr>
<td>AArch32</td>
<td>DCIMVAC4</td>
<td>Data Cache line Clean by VA to PoC</td>
</tr>
<tr>
<td>AArch32</td>
<td>DCMVAC4</td>
<td>Data Cache line Clean by VA to PoU</td>
</tr>
<tr>
<td>AArch32</td>
<td>DCISW</td>
<td>Data Cache line Invalidate by Set/Way</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICIALLU</td>
<td>Instruction Cache Invalidate All to PoU</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICIALLUS</td>
<td>Instruction Cache Invalidate All to PoU, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICMVAC4</td>
<td>Data Cache line Invalidate by VA to PoU</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC GDSW</td>
<td>Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC GDVAC</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC GDSW</td>
<td>Data, Allocation Tag or unified Cache line Clean of Data and Allocation Tags by VA to PoDP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PDP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoU</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoDP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoU</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CGVAP</td>
<td>Data, Allocation Tag or unified Cache line Clean of Allocation Tags by VA to PoU</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CIVAC</td>
<td>Data, Allocation Tag or unified Cache line Clean by VA to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CSW</td>
<td>Data or unified Cache line Clean by Set/Way</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CVAC</td>
<td>Data or unified Cache line Clean by VA to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CVADP</td>
<td>Data or unified Cache line Clean by VA to PoDP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CVAP</td>
<td>Data or unified Cache line Clean by VA to PoP</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC CVAU</td>
<td>Data or unified Cache line Clean by VA to PoU</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC GVA</td>
<td>Data Cache set Allocation Tag by VA</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC GZVA</td>
<td>Data Cache set Allocation Tags and Zero by VA</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC IGDSW</td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Data and Allocation Tags by Set/Way</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC IGDVAC</td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by VA to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC IGDSW</td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Va to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC IGDSW</td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Va to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC IGDSW</td>
<td>Data, Allocation Tag or unified Cache line Invalidate of Allocation Tags by Va to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC ISW</td>
<td>Data or unified Cache line Invalidate by Set/Way</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC ISW</td>
<td>Data or unified Cache line Invalidate by VA to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC IVAC</td>
<td>Data or unified Cache line Invalidate by VA to PoC</td>
</tr>
<tr>
<td>AArch64</td>
<td>DC ZVA</td>
<td>Data Cache Zero by VA</td>
</tr>
<tr>
<td>AArch64</td>
<td>IC IALLL</td>
<td>Instruction Cache Invalidate All to PoU</td>
</tr>
<tr>
<td>AArch64</td>
<td>IC IALLLIS</td>
<td>Instruction Cache Invalidate All to PoU, Inner Shareable</td>
</tr>
</tbody>
</table>
## In the Address functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>ATS12NSOPR</td>
<td>Address Translate Stages 1 and 2 Non-secure Only PL1 Read</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS12NSOPW</td>
<td>Address Translate Stages 1 and 2 Non-secure Only PL1 Write</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS12NSOUR</td>
<td>Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS12NSOUW</td>
<td>Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1CPR</td>
<td>Address Translate Stage 1 Current state PL1 Read</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1CPRP</td>
<td>Address Translate Stage 1 Current state PL1 Read PAN</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1CPUW</td>
<td>Address Translate Stage 1 Current state PL1 Write</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1CUR</td>
<td>Address Translate Stage 1 Current state Unprivileged Read</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1CUW</td>
<td>Address Translate Stage 1 Current state Unprivileged Write</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1HR</td>
<td>Address Translate Stage 1 Hyp mode Read</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1HW</td>
<td>Address Translate Stage 1 Hyp mode Write</td>
</tr>
<tr>
<td>AArch32</td>
<td>PAR</td>
<td>Physical Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S12E0R</td>
<td>Address Translate Stages 1 and 2 EL0 Read</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S12E0W</td>
<td>Address Translate Stages 1 and 2 EL0 Write</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S12E1R</td>
<td>Address Translate Stages 1 and 2 EL1 Read</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S12E1W</td>
<td>Address Translate Stages 1 and 2 EL1 Write</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E0R</td>
<td>Address Translate Stage 1 EL0 Read</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E0W</td>
<td>Address Translate Stage 1 EL0 Write</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E1R</td>
<td>Address Translate Stage 1 EL1 Read</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E1WP</td>
<td>Address Translate Stage 1 EL1 Write PAN</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E2R</td>
<td>Address Translate Stage 1 EL2 Read</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E2W</td>
<td>Address Translate Stage 1 EL2 Write</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E3R</td>
<td>Address Translate Stage 1 EL3 Read</td>
</tr>
<tr>
<td>AArch64</td>
<td>AT S1E3W</td>
<td>Address Translate Stage 1 EL3 Write</td>
</tr>
<tr>
<td>AArch64</td>
<td>PAR_ELI</td>
<td>Physical Address Register</td>
</tr>
</tbody>
</table>

## In the TLB functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>CFPRCTX</td>
<td>Control Flow Prediction Restriction by Context</td>
</tr>
<tr>
<td>AArch32</td>
<td>CPPRCTX</td>
<td>Cache Prefetch Prediction Restriction by Context</td>
</tr>
<tr>
<td>AArch32</td>
<td>DTLBIALL</td>
<td>Data TLB Invalidate All</td>
</tr>
<tr>
<td>AArch32</td>
<td>DTLBIASID</td>
<td>Data TLB Invalidate by ASID match</td>
</tr>
<tr>
<td>AArch32</td>
<td>DTLBIMVA</td>
<td>Data TLB Invalidate by VA</td>
</tr>
<tr>
<td>AArch32</td>
<td>DVPRCTX</td>
<td>Data Value Prediction Restriction by Context</td>
</tr>
<tr>
<td>AArch32</td>
<td>ITLBIALL</td>
<td>Instruction TLB Invalidate All</td>
</tr>
<tr>
<td>AArch32</td>
<td>ITLBIASID</td>
<td>Instruction TLB Invalidate by ASID match</td>
</tr>
<tr>
<td>AArch32</td>
<td>ITLBIMVA</td>
<td>Instruction TLB Invalidate by VA</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVA</td>
<td>TLB Invalidate by VA</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMAA</td>
<td>TLB Invalidate by VA, All ASID</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVA</td>
<td>TLB Invalidate by VA</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAA</td>
<td>TLB Invalidate by VA, All ASID</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAAIS</td>
<td>TLB Invalidate by VA, All ASID, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAAL</td>
<td>TLB Invalidate by VA, All ASID, Last level</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAALIS</td>
<td>TLB Invalidate by VA, All ASID, Last level, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAH</td>
<td>TLB Invalidate by VA, Hyp mode</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAHIS</td>
<td>TLB Invalidate by VA, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAIS</td>
<td>TLB Invalidate by VA, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAL</td>
<td>TLB Invalidate by VA, Last level</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVALH</td>
<td>TLB Invalidate by VA, Last level, Hyp mode</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVALHIS</td>
<td>TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVALIS</td>
<td>TLB Invalidate by VA, Last level, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE1</td>
<td>TLB Invalidate All, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE1IS</td>
<td>TLB Invalidate All, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE1OS</td>
<td>TLB Invalidate All, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE2</td>
<td>TLB Invalidate All, EL2</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE2IS</td>
<td>TLB Invalidate All, EL2, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE2OS</td>
<td>TLB Invalidate All, EL2, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE3</td>
<td>TLB Invalidate All, EL3</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE3IS</td>
<td>TLB Invalidate All, EL3, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ALLE3OS</td>
<td>TLB Invalidate All, EL3, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ASIDE1</td>
<td>TLB Invalidate by ASID, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ASIDE1IS</td>
<td>TLB Invalidate by ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI ASIDE1OS</td>
<td>TLB Invalidate by ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI IPAS2E1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI IPAS2E1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI IPAS2E1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI IPAS2LE1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI IPAS2LE1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI IPAS2LE1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RIPAS2E1</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RIPAS2E1IS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RIPAS2E1OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RIPAS2LE1</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RIPAS2LE1IS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RIPAS2LE1OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVAAE1</td>
<td>TLB Range Invalidate by VA, All ASID, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVAAE1IS</td>
<td>TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVAAE1OS</td>
<td>TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVAALE1</td>
<td>TLB Range Invalidate by VA, All ASID, Last level, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVAALE1IS</td>
<td>TLB Range Invalidate by VA, All ASID, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVAALE1OS</td>
<td>TLB Range Invalidate by VA, All ASID, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE1</td>
<td>TLB Range Invalidate by VA, Last level, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE1IS</td>
<td>TLB Range Invalidate by VA, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE1OS</td>
<td>TLB Range Invalidate by VA, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE2</td>
<td>TLB Range Invalidate by VA, EL2</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE2IS</td>
<td>TLB Range Invalidate by VA, EL2, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE2OS</td>
<td>TLB Range Invalidate by VA, EL2, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE3</td>
<td>TLB Range Invalidate by VA, EL3</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE3IS</td>
<td>TLB Range Invalidate by VA, EL3, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI RVALE3OS</td>
<td>TLB Range Invalidate by VA, EL3, Outer Shareable</td>
</tr>
</tbody>
</table>
### System Register index by functional group

#### TLB Invalidation

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>TLBI VAAE1</td>
<td>TLB Invalidate by VA, All ASID, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAAE1IS</td>
<td>TLB Invalidate by VA, All ASID, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAAE1OS</td>
<td>TLB Invalidate by VA, All ASID, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE1</td>
<td>TLB Invalidate by VA, All ASID, Last level, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE1IS</td>
<td>TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE1OS</td>
<td>TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE1</td>
<td>TLB Invalidate by VA, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE1IS</td>
<td>TLB Invalidate by VA, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE1OS</td>
<td>TLB Invalidate by VA, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE2</td>
<td>TLB Invalidate by VA, EL2</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE2IS</td>
<td>TLB Invalidate by VA, EL2, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE2OS</td>
<td>TLB Invalidate by VA, EL2, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE3</td>
<td>TLB Invalidate by VA, EL3</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE3IS</td>
<td>TLB Invalidate by VA, EL3, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VAE3OS</td>
<td>TLB Invalidate by VA, EL3, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE1</td>
<td>TLB Invalidate by VA, Last level, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE1IS</td>
<td>TLB Invalidate by VA, Last level, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE1OS</td>
<td>TLB Invalidate by VA, Last level, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE2</td>
<td>TLB Invalidate by VA, Last level, EL2</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE2IS</td>
<td>TLB Invalidate by VA, Last level, EL2, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE2OS</td>
<td>TLB Invalidate by VA, Last level, EL2, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE3</td>
<td>TLB Invalidate by VA, Last level, EL3</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE3IS</td>
<td>TLB Invalidate by VA, Last level, EL3, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VALE3OS</td>
<td>TLB Invalidate by VA, Last level, EL3, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VMALLE1</td>
<td>TLB Invalidate by VMID, All at stage 1, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VMALLE1IS</td>
<td>TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBI VMALLE1OS</td>
<td>TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI VMALLS12E1</td>
<td>TLB Invalidate by VMID, All at Stage 1 and 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI VMALLS12E1IS</td>
<td>TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLI VMALLS12E1OS</td>
<td>TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable</td>
</tr>
</tbody>
</table>

#### PMU (Performance Monitors Unit)

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>PMCCFILTR</td>
<td>Performance Monitors Cycle Count Filter Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCCNTR</td>
<td>Performance Monitors Cycle Count Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCEID0</td>
<td>Performance Monitors Common Event Identification register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCEID1</td>
<td>Performance Monitors Common Event Identification register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCEID2</td>
<td>Performance Monitors Common Event Identification register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCEID3</td>
<td>Performance Monitors Common Event Identification register 3</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCNTENCLR</td>
<td>Performance Monitors Count Enable Clear register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCNTENSET</td>
<td>Performance Monitors Count Enable Set register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMCR</td>
<td>Performance Monitors Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMEVCNTR&lt;n&gt;</td>
<td>Performance Monitors Event Count Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMEVTYPER&lt;n&gt;</td>
<td>Performance Monitors Event Type Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMINTENCLR</td>
<td>Performance Monitors Interrupt Enable Clear register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMINTENSET</td>
<td>Performance Monitors Interrupt Enable Set register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMMIR</td>
<td>Performance Monitors Machine Identification Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMOVSR</td>
<td>Performance Monitors Overflow Flag Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMOVSSET</td>
<td>Performance Monitors Overflow Flag Status Set register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMSELR</td>
<td>Performance Monitors Event Counter Selection Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMSWINC</td>
<td>Performance Monitors Software Increment register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMUSERENR</td>
<td>Performance Monitors User Enable Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMXEVCTR</td>
<td>Performance Monitors Selected Event Count Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>PMXEVTPER</td>
<td>Performance Monitors Selected Event Type Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMCCFILTR_EL0</td>
<td>Performance Monitors Cycle Count Filter Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMCCNTR_EL0</td>
<td>Performance Monitors Cycle Count Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMCEID0_EL0</td>
<td>Performance Monitors Common Event Identification register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMCEID1_EL0</td>
<td>Performance Monitors Common Event Identification register 1</td>
</tr>
</tbody>
</table>

In the PMU functional group:
<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>PMCNTENCLR_EL0</td>
<td>Performance Monitors Count Enable Clear register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMCNTENSET_EL0</td>
<td>Performance Monitors Count Enable Set register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMCR_EL0</td>
<td>Performance Monitors Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMEVCNTR&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Count Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMEVENTYPE&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Type Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMINTENCLR_EL1</td>
<td>Performance Monitors Interrupt Enable Clear register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMINTENSET_EL1</td>
<td>Performance Monitors Interrupt Enable Set register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMISR_EL1</td>
<td>Performance Monitors Machine Identification Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMOVSCLR_EL0</td>
<td>Performance Monitors Overflow Flag Status Clear Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMOVSSET_EL0</td>
<td>Performance Monitors Overflow Flag Status Set register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMSELR_EL0</td>
<td>Performance Monitors Event Counter Selection Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMSWINC_EL0</td>
<td>Performance Monitors Software Increment register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMUSERENR_EL0</td>
<td>Performance Monitors User Enable Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMXEVCTR_EL0</td>
<td>Performance Monitors Selected Event Count Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>PMXEVENTYPE_EL0</td>
<td>Performance Monitors Selected Event Type Register</td>
</tr>
<tr>
<td>External</td>
<td>PMAUTHSTATUS</td>
<td>Performance Monitors Authentication Status register</td>
</tr>
<tr>
<td>External</td>
<td>PMCCFILTR_EL0</td>
<td>Performance Monitors Cycle Counter Filter Register</td>
</tr>
<tr>
<td>External</td>
<td>PMCCNTN_EL0</td>
<td>Performance Monitors Cycle Counter</td>
</tr>
<tr>
<td>External</td>
<td>PMCID0</td>
<td>Performance Monitors Common Event Identification register 0</td>
</tr>
<tr>
<td>External</td>
<td>PMCID1</td>
<td>Performance Monitors Common Event Identification register 1</td>
</tr>
<tr>
<td>External</td>
<td>PMCID2</td>
<td>Performance Monitors Common Event Identification register 2</td>
</tr>
<tr>
<td>External</td>
<td>PMCID3</td>
<td>Performance Monitors Common Event Identification register 3</td>
</tr>
<tr>
<td>External</td>
<td>PMCFGR</td>
<td>Performance Monitors Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>PMCID1SR</td>
<td>CONTEXTIDR_EL1 Sample Register</td>
</tr>
<tr>
<td>External</td>
<td>PMCID2SR</td>
<td>CONTEXTIDR_EL2 Sample Register</td>
</tr>
<tr>
<td>External</td>
<td>PMCIDR0</td>
<td>Performance Monitors Component Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>PMCIDR1</td>
<td>Performance Monitors Component Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>PMCIDR2</td>
<td>Performance Monitors Component Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>PMCIDR3</td>
<td>Performance Monitors Component Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>PMCNTENCLR_EL0</td>
<td>Performance Monitors Count Enable Clear register</td>
</tr>
<tr>
<td>External</td>
<td>PMINTENSET_EL1</td>
<td>Performance Monitors Interrupt Enable Set register</td>
</tr>
<tr>
<td>External</td>
<td>PMCR_EL0</td>
<td>Performance Monitors Control Register</td>
</tr>
<tr>
<td>External</td>
<td>PMDEVAFF0</td>
<td>Performance Monitors Device Affinity register 0</td>
</tr>
<tr>
<td>External</td>
<td>PMDEVAFF1</td>
<td>Performance Monitors Device Affinity register 1</td>
</tr>
<tr>
<td>External</td>
<td>PMDEVARCH</td>
<td>Performance Monitors Device Architecture register</td>
</tr>
<tr>
<td>External</td>
<td>PMDEVID</td>
<td>Performance Monitors Device ID register</td>
</tr>
<tr>
<td>External</td>
<td>PMDEVTYPE</td>
<td>Performance Monitors Device Type register</td>
</tr>
<tr>
<td>External</td>
<td>PMEVCNTR&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Count Registers</td>
</tr>
<tr>
<td>External</td>
<td>PMEVENTYPE&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Type Registers</td>
</tr>
<tr>
<td>External</td>
<td>PMINTENCLR_EL1</td>
<td>Performance Monitors Interrupt Enable Clear register</td>
</tr>
<tr>
<td>External</td>
<td>PMINTENSEL_EL1</td>
<td>Performance Monitors Interrupt Enable Set register</td>
</tr>
<tr>
<td>External</td>
<td>PMITCTRL</td>
<td>Performance Monitors Integration mode Control Register</td>
</tr>
<tr>
<td>External</td>
<td>PMLAR</td>
<td>Performance Monitors Lock Access Register</td>
</tr>
<tr>
<td>External</td>
<td>PMLSR</td>
<td>Performance Monitors Lock Status Register</td>
</tr>
<tr>
<td>External</td>
<td>PMMIR</td>
<td>Performance Monitors Machine Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>PMOVSCLR_EL0</td>
<td>Performance Monitors Overflow Flag Status Clear register</td>
</tr>
<tr>
<td>External</td>
<td>PMOVSSET_EL0</td>
<td>Performance Monitors Overflow Flag Status Set register</td>
</tr>
<tr>
<td>External</td>
<td>PMPCSR</td>
<td>Program Counter Sample Register</td>
</tr>
<tr>
<td>External</td>
<td>PMPIDR0</td>
<td>Performance Monitors Peripheral Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>PMPIDR1</td>
<td>Performance Monitors Peripheral Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>PMPIDR2</td>
<td>Performance Monitors Peripheral Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>PMPIDR3</td>
<td>Performance Monitors Peripheral Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>PMPIDR4</td>
<td>Performance Monitors Peripheral Identification Register 4</td>
</tr>
<tr>
<td>External</td>
<td>PMSWINC_EL0</td>
<td>Performance Monitors Software Increment register</td>
</tr>
<tr>
<td>External</td>
<td>PMVIDSР</td>
<td>VMID Sample Register</td>
</tr>
</tbody>
</table>

In the Reset functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>HRMR</td>
<td>Hyp Reset Management Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>RMR</td>
<td>Reset Management Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>RVBAR</td>
<td>Reset Vector Base Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>BMR_EL1</td>
<td>Reset Management Register (EL1)</td>
</tr>
</tbody>
</table>
### System Register index by functional group

#### Exec state | Name | Description
--- | --- | ---
AArch64 | **RMR_EL2** | Reset Management Register (EL2)
AArch64 | **RMR_EL3** | Reset Management Register (EL3)
AArch64 | **RVBAR_EL1** | Reset Vector Base Address Register (if EL2 and EL3 not implemented)
AArch64 | **RVBAR_EL2** | Reset Vector Base Address Register (if EL3 not implemented)
AArch64 | **RVBAR_EL3** | Reset Vector Base Address Register (if EL3 implemented)

**In the Thread functional group:**

#### Exec state | Name | Description
--- | --- | ---
AArch32 | **HTPIDR** | Hyp Software Thread ID Register
AArch32 | **TPIDRPRW** | PL1 Software Thread ID Register
AArch32 | **TPIDRURO** | PL0 Read-Only Software Thread ID Register
AArch32 | **TPIDRURW** | PL0 Read/Write Software Thread ID Register
AArch64 | **SCXTNUM_EL0** | EL0 Read/Write Software Context Number
AArch64 | **SCXTNUM_EL1** | EL1 Read/Write Software Context Number
AArch64 | **SCXTNUM_EL2** | EL2 Read/Write Software Context Number
AArch64 | **SCXTNUM_EL3** | EL3 Read/Write Software Context Number
AArch64 | **TPIDRRO_EL0** | EL0 Read-Only Software Thread ID Register
AArch64 | **TPIDR_EL0** | EL0 Read/Write Software Thread ID Register
AArch64 | **TPIDR_EL1** | EL1 Software Thread ID Register
AArch64 | **TPIDR_EL2** | EL2 Software Thread ID Register
AArch64 | **TPIDR_EL3** | EL3 Software Thread ID Register

**In the IMP DEF functional group:**

#### Exec state | Name | Description
--- | --- | ---
AArch32 | **ACTLR** | Auxiliary Control Register
AArch32 | **ACTLR2** | Auxiliary Control Register 2
AArch32 | **ADFSR** | Auxiliary Data Fault Status Register
AArch32 | **AIDR** | Auxiliary ID Register
AArch32 | **AIFSR** | Auxiliary Instruction Fault Status Register
AArch32 | **AMAIR0** | Auxiliary Memory Attribute Indirection Register 0
AArch32 | **AMAIR1** | Auxiliary Memory Attribute Indirection Register 1
AArch32 | **HACTLR** | Hyp Auxiliary Control Register
AArch32 | **HACTLR2** | Hyp Auxiliary Control Register 2
AArch32 | **HADFSR** | Hyp Auxiliary Data Fault Status Register
AArch32 | **HAIFSR** | Hyp Auxiliary Instruction Fault Status Register
AArch32 | **HAMAIR0** | Hyp Auxiliary Memory Attribute Indirection Register 0
AArch32 | **HAMAIR1** | Hyp Auxiliary Memory Attribute Indirection Register 1
AArch64 | **ACTLR_EL1** | Auxiliary Control Register (EL1)
AArch64 | **ACTLR_EL2** | Auxiliary Control Register (EL2)
AArch64 | **ACTLR_EL3** | Auxiliary Control Register (EL3)
AArch64 | **AFSR0_EL1** | Auxiliary Fault Status Register 0 (EL1)
AArch64 | **AFSR0_EL2** | Auxiliary Fault Status Register 0 (EL2)
AArch64 | **AFSR0_EL3** | Auxiliary Fault Status Register 0 (EL3)
AArch64 | **AFSR1_EL1** | Auxiliary Fault Status Register 1 (EL1)
AArch64 | **AFSR1_EL2** | Auxiliary Fault Status Register 1 (EL2)
AArch64 | **AFSR1_EL3** | Auxiliary Fault Status Register 1 (EL3)
AArch64 | **AIDR_EL1** | Auxiliary ID Register
AArch64 | **AMAIR_EL1** | Auxiliary Memory Attribute Indirection Register (EL1)
AArch64 | **AMAIR_EL2** | Auxiliary Memory Attribute Indirection Register (EL2)
AArch64 | **AMAIR_EL3** | Auxiliary Memory Attribute Indirection Register (EL3)
AArch64 | **HACR_EL2** | Hypervisor Auxiliary Control Register
AArch64 | **S1 <op1> <Cn> <Cm> <op2>** | IMPLEMENTATION DEFINED maintenance instructions
AArch64 | **S3 <op1> <Cn> <Cm> <op2>** | IMPLEMENTATION DEFINED registers

**In the Timer functional group:**

#### Exec state | Name | Description
--- | --- | ---
AArch32 | **CNTFRQ** | Counter-timer Frequency register
AArch32 | **CNTHPS_CTL** | Counter-timer Secure Physical Timer Control Register (EL2)
<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>CNTHPS.CVVAL</td>
<td>Counter-timer Secure Physical Timer CompareValue Register (EL2)</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHPS.TVVAL</td>
<td>Counter-timer Secure Physical Timer TimerValue Register (EL2)</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHPS.CTL</td>
<td>Counter-timer Hyp Physical Timer Control register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHVS.CVVAL</td>
<td>Counter-timer Secure Virtual Timer CompareValue Register (EL2)</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHVS.TVVAL</td>
<td>Counter-timer Secure Virtual Timer TimerValue Register (EL2)</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHV.CTL</td>
<td>Counter-timer Virtual Timer Control register (EL2)</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHV.CVVAL</td>
<td>Counter-timer Virtual Timer CompareValue register (EL2)</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHV.TVVAL</td>
<td>Counter-timer Virtual Timer TimerValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHVS.CVVAL.EL2</td>
<td>Counter-timer Secure Virtual Timer CompareValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHVS.TVVAL.EL2</td>
<td>Counter-timer Secure Virtual Timer TimerValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHV.CTL.EL2</td>
<td>Counter-timer Virtual Timer Control register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHV.CVVAL.EL2</td>
<td>Counter-timer Virtual Timer CompareValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHV.TVVAL.EL2</td>
<td>Counter-timer Virtual Timer TimerValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.EL0</td>
<td>Counter-timer Frequency register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.CTSS.EL0</td>
<td>Counter-timer Self-Synchronized Physical Count register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.CT.EL0</td>
<td>Counter-timer Physical Count register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.OFF.EL0</td>
<td>Counter-timer Physical Offset register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTPS.CTL.EL1</td>
<td>Counter-timer Physical Secure Timer Control register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTPS.CVVAL.EL1</td>
<td>Counter-timer Physical Secure Timer CompareValue register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTPS.TVVAL.EL1</td>
<td>Counter-timer Physical Secure Timer TimerValue register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.EL0</td>
<td>Counter-timer Physical Timer Control register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.CVVAL.EL0</td>
<td>Counter-timer Physical Timer CompareValue register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTP.TVVAL.EL0</td>
<td>Counter-timer Physical Timer TimerValue register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTV.CTSS.EL0</td>
<td>Counter-timer Self-Synchronized Virtual Count register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTV.EL0</td>
<td>Counter-timer Virtual Count register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTV.CVVAL.EL0</td>
<td>Counter-timer Virtual Timer Control register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTV.CVVAL.EL0</td>
<td>Counter-timer Virtual Timer CompareValue register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTV.TVVAL.EL0</td>
<td>Counter-timer Virtual Timer TimerValue register</td>
</tr>
<tr>
<td>External</td>
<td>CNTACR&lt;n&gt;</td>
<td>Counter-timer Access Control Registers</td>
</tr>
<tr>
<td>External</td>
<td>CNTCR</td>
<td>Counter Control Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTCV</td>
<td>Counter Count Value register</td>
</tr>
<tr>
<td>External</td>
<td>CNTEL0ACR</td>
<td>Counter-timer EL0 Access Control Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTFID0</td>
<td>Counter Frequency ID</td>
</tr>
<tr>
<td>External</td>
<td>CNTFID&lt;n&gt;</td>
<td>Counter Frequency IDs, n &gt; 0</td>
</tr>
<tr>
<td>External</td>
<td>CNTFROI</td>
<td>Counter-timer Frequency Control</td>
</tr>
<tr>
<td>External</td>
<td>CNTID</td>
<td>Counter Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTP.SAR</td>
<td>Counter-timer Non-secure Access Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTP.CT</td>
<td>Counter-timer Physical Count</td>
</tr>
<tr>
<td>External</td>
<td>CNTP.CTL</td>
<td>Counter-timer Physical Timer Control</td>
</tr>
<tr>
<td>External</td>
<td>CNTP.CVVAL</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>External</td>
<td>CNTP.TVVAL</td>
<td>Counter-timer Physical Timer TimerValue</td>
</tr>
<tr>
<td>External</td>
<td>CNT.SCR</td>
<td>Counter Scale Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTSR</td>
<td>Counter Status Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTTIDR</td>
<td>Counter-timer Timer ID Register</td>
</tr>
<tr>
<td>External</td>
<td>CNTV.CT</td>
<td>Counter-timer Virtual Count</td>
</tr>
<tr>
<td>External</td>
<td>CNTV.OFF</td>
<td>Counter-timer Virtual Offset</td>
</tr>
<tr>
<td>External</td>
<td>CNTV.OFF&lt;n&gt;</td>
<td>Counter-timer Virtual Offsets</td>
</tr>
<tr>
<td>External</td>
<td>CNTV.CTL</td>
<td>Counter-timer Virtual Timer Control</td>
</tr>
<tr>
<td>External</td>
<td>CNTV.CVVAL</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>External</td>
<td>CNTV.TVVAL</td>
<td>Counter-timer Virtual Timer TimerValue</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>External</td>
<td>CounterID&lt;n&gt;</td>
<td>Counter ID registers</td>
</tr>
</tbody>
</table>

In the Debug functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>DBGAUTHSTATUS</td>
<td>Debug Authentication Status register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGBCR&lt;n&gt;</td>
<td>Debug Breakpoint Control Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGBVVR&lt;n&gt;</td>
<td>Debug Breakpoint Value Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGBXVR&lt;n&gt;</td>
<td>Debug Breakpoint Extended Value Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGCLAIMCLR</td>
<td>Debug CLAIM Tag Clear register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGCLAIMSET</td>
<td>Debug CLAIM Tag Set register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDCCINT</td>
<td>DCC Interrupt Enable Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDEVID</td>
<td>Debug Device ID register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDEVID1</td>
<td>Debug Device ID register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDEVID2</td>
<td>Debug Device ID register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDIR</td>
<td>Debug ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDRAR</td>
<td>Debug ROM Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDSAR</td>
<td>Debug Self Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDSCRxExt</td>
<td>Debug Status and Control Register, External View</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDCRxInt</td>
<td>Debug Status and Control Register, Internal View</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDTRRXext</td>
<td>Debug OS Lock Data Transfer Register, Receive, External View</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDTRRXint</td>
<td>Debug Data Transfer Register, Receive</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDTRTXext</td>
<td>Debug OS Lock Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGDTRTXint</td>
<td>Debug Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGOSDLR</td>
<td>Debug OS Double Lock Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGOSECCR</td>
<td>Debug OS Lock Exception Catch Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGOSLAR</td>
<td>Debug OS Lock Access Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGOSLAR</td>
<td>Debug OS Lock Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGPCR</td>
<td>Debug Power Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGVCR</td>
<td>Debug Vector Catch Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGWCR&lt;n&gt;</td>
<td>Debug Watchpoint Control Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGWFAR</td>
<td>Debug Watchpoint Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>DBGWVR&lt;n&gt;</td>
<td>Debug Watchpoint Value Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>TRFCR</td>
<td>Trace Filter Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGAUTHSTATUS_EL1</td>
<td>Debug Authentication Status register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGBCR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Control Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGBVVR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Value Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGCLAIMCLR_EL1</td>
<td>Debug CLAIM Tag Clear register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGCLAIMSET_EL1</td>
<td>Debug CLAIM Tag Set register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGDTRRX_EL0</td>
<td>Debug Data Transfer Register, Receive</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGDTRTX_EL0</td>
<td>Debug Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGDTR_EL0</td>
<td>Debug Data Transfer Register, half-duplex</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGPCR_EL1</td>
<td>Debug Power Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGVCR32_EL2</td>
<td>Debug Vector Catch Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGWCR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Control Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>DBGWVR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Value Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>DLR_EL0</td>
<td>Debug Link Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>DSPSR_EL0</td>
<td>Debug Saved Program Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>MDCCINT_EL1</td>
<td>Monitor DCC Interrupt Enable Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>MDCCSR_EL0</td>
<td>Monitor DCC Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>MDRAR_EL1</td>
<td>Monitor Debug ROM Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>MDSR_EL1</td>
<td>Monitor Debug System Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>OSDLR_EL1</td>
<td>OS Double Lock Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>OSDTRRX_EL1</td>
<td>OS Lock Data Transfer Register, Receive</td>
</tr>
<tr>
<td>AArch64</td>
<td>OSDTRTX_EL1</td>
<td>OS Lock Data Transfer Register, Transit</td>
</tr>
<tr>
<td>AArch64</td>
<td>OSECCL_EL1</td>
<td>OS Lock Exception Catch Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>OSLAR_EL1</td>
<td>OS Lock Access Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>OSI_SR_EL1</td>
<td>OS Lock Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRFCR_EL1</td>
<td>Trace Filter Control Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRFCR_EL2</td>
<td>Trace Filter Control Register (EL2)</td>
</tr>
<tr>
<td>External</td>
<td>DBGAUTHSTATUS_EL1</td>
<td>Debug Authentication Status register</td>
</tr>
<tr>
<td>External</td>
<td>DBGBCR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Control Registers</td>
</tr>
<tr>
<td>External</td>
<td>DBGBVVR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Value Registers</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>External</td>
<td>DBGCLAIMCLR_EL1</td>
<td>Debug CLAIM Tag Clear register</td>
</tr>
<tr>
<td>External</td>
<td>DBGCLAIMSET_EL1</td>
<td>Debug CLAIM Tag Set register</td>
</tr>
<tr>
<td>External</td>
<td>DBGDTTRX_EL0</td>
<td>Debug Data Transfer Register, Receive</td>
</tr>
<tr>
<td>External</td>
<td>DBGDTTRX_EL0</td>
<td>Debug Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>External</td>
<td>DBGWCR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Control Registers</td>
</tr>
<tr>
<td>External</td>
<td>DBGWVR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Value Registers</td>
</tr>
<tr>
<td>External</td>
<td>EDACR</td>
<td>External Debug Auxiliary Control Register</td>
</tr>
<tr>
<td>External</td>
<td>EDCIDR0</td>
<td>External Debug Component Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>EDCIDR1</td>
<td>External Debug Component Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>EDCIDR2</td>
<td>External Debug Component Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>EDCIDR3</td>
<td>External Debug Component Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>EDCIDSR</td>
<td>External Debug Context ID Sample Register</td>
</tr>
<tr>
<td>External</td>
<td>EDDAVAFF0</td>
<td>External Debug Device Affinity register 0</td>
</tr>
<tr>
<td>External</td>
<td>EDDAVAFF1</td>
<td>External Debug Device Affinity register 1</td>
</tr>
<tr>
<td>External</td>
<td>EDDAVARCH</td>
<td>External Debug Device Architecture register</td>
</tr>
<tr>
<td>External</td>
<td>EDEVID</td>
<td>External Debug Device ID register 0</td>
</tr>
<tr>
<td>External</td>
<td>EDEVID1</td>
<td>External Debug Device ID register 1</td>
</tr>
<tr>
<td>External</td>
<td>EDEVID2</td>
<td>External Debug Device ID register 2</td>
</tr>
<tr>
<td>External</td>
<td>EDEVTYPF</td>
<td>External Debug Device Type register</td>
</tr>
<tr>
<td>External</td>
<td>EDECGR</td>
<td>External Debug Exception Catch Control Register</td>
</tr>
<tr>
<td>External</td>
<td>EDECR</td>
<td>External Debug Execution Control Register</td>
</tr>
<tr>
<td>External</td>
<td>EDESRS</td>
<td>External Debug Event Status Register</td>
</tr>
<tr>
<td>External</td>
<td>EDITCTRL</td>
<td>External Debug Integration mode Control register</td>
</tr>
<tr>
<td>External</td>
<td>EDITR</td>
<td>External Debug Instruction Transfer Register</td>
</tr>
<tr>
<td>External</td>
<td>EDLAR</td>
<td>External Debug Lock Access Register</td>
</tr>
<tr>
<td>External</td>
<td>EDLSR</td>
<td>External Debug Lock Status Register</td>
</tr>
<tr>
<td>External</td>
<td>EDPCSR</td>
<td>External Debug Program Counter Sample Register</td>
</tr>
<tr>
<td>External</td>
<td>EDPIDR0</td>
<td>External Debug Peripheral Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>EDPIDR1</td>
<td>External Debug Peripheral Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>EDPIDR2</td>
<td>External Debug Peripheral Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>EDPIDR3</td>
<td>External Debug Peripheral Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>EDPIDR4</td>
<td>External Debug Peripheral Identification Register 4</td>
</tr>
<tr>
<td>External</td>
<td>EDPRCR</td>
<td>External Debug Power/Reset Control Register</td>
</tr>
<tr>
<td>External</td>
<td>EDPRSR</td>
<td>External Debug Processor Status Register</td>
</tr>
<tr>
<td>External</td>
<td>EDRCR</td>
<td>External Debug Reserve Control Register</td>
</tr>
<tr>
<td>External</td>
<td>EDSCR</td>
<td>External Debug Status and Control Register</td>
</tr>
<tr>
<td>External</td>
<td>EDVIDSRS</td>
<td>External Debug Virtual Context Sample Register</td>
</tr>
<tr>
<td>External</td>
<td>EDWAR</td>
<td>External Debug Watchpoint Address Register</td>
</tr>
<tr>
<td>External</td>
<td>OSLAR_EL1</td>
<td>OS Lock Access Register</td>
</tr>
</tbody>
</table>

In the CTI functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>ASICCTL</td>
<td>CTI External Multiplexer Control register</td>
</tr>
<tr>
<td>External</td>
<td>CTIAPPCLEAR</td>
<td>CTI Application Trigger Clear register</td>
</tr>
<tr>
<td>External</td>
<td>CTIAPPULSE</td>
<td>CTI Application Pulse register</td>
</tr>
<tr>
<td>External</td>
<td>CTIAPPSET</td>
<td>CTI Application Trigger Set register</td>
</tr>
<tr>
<td>External</td>
<td>CTIAUTHSTATUS</td>
<td>CTI Authentication Status register</td>
</tr>
<tr>
<td>External</td>
<td>CTICHINSTATUS</td>
<td>CTI Channel In Status register</td>
</tr>
<tr>
<td>External</td>
<td>CTICHOUTSTATUS</td>
<td>CTI Channel Out Status register</td>
</tr>
<tr>
<td>External</td>
<td>CTICIDR0</td>
<td>CTI Component Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>CTICIDR1</td>
<td>CTI Component Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>CTICIDR2</td>
<td>CTI Component Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>CTICIDR3</td>
<td>CTI Component Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>CTICLAIMCLR</td>
<td>CTI CLAIM Tag Clear register</td>
</tr>
<tr>
<td>External</td>
<td>CTICLAIMSET</td>
<td>CTI CLAIM Tag Set register</td>
</tr>
<tr>
<td>External</td>
<td>CTICONTROL</td>
<td>CTI Control register</td>
</tr>
<tr>
<td>External</td>
<td>CTIDAVAFF0</td>
<td>CTI Device Affinity register 0</td>
</tr>
<tr>
<td>External</td>
<td>CTIDAVAFF1</td>
<td>CTI Device Affinity register 1</td>
</tr>
<tr>
<td>External</td>
<td>CTIDAVARCH</td>
<td>CTI Device Architecture register</td>
</tr>
<tr>
<td>External</td>
<td>CTIDEVCTRL</td>
<td>CTI Device Control register</td>
</tr>
<tr>
<td>External</td>
<td>CTIDEVID</td>
<td>CTI Device ID register 0</td>
</tr>
<tr>
<td>External</td>
<td>CTIDEVID1</td>
<td>CTI Device ID register 1</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>External</td>
<td>CTIDEVID2</td>
<td>CTI Device ID register 2</td>
</tr>
<tr>
<td>External</td>
<td>CTIDEVTYPE</td>
<td>CTI Device Type register</td>
</tr>
<tr>
<td>External</td>
<td>CTIGATE</td>
<td>CTI Channel Gate Enable register</td>
</tr>
<tr>
<td>External</td>
<td>CTHINEN&lt;n&gt;</td>
<td>CTI Input Trigger to Output Channel Enable registers</td>
</tr>
<tr>
<td>External</td>
<td>CTHINTACK</td>
<td>CTI Output Trigger Acknowledge register</td>
</tr>
<tr>
<td>External</td>
<td>CTHITCTRL</td>
<td>CTI Integration mode Control register</td>
</tr>
<tr>
<td>External</td>
<td>CTHILAR</td>
<td>CTI Lock Access Register</td>
</tr>
<tr>
<td>External</td>
<td>CTHILSR</td>
<td>CTI Lock Status Register</td>
</tr>
<tr>
<td>External</td>
<td>CTIOUTEN&lt;n&gt;</td>
<td>CTI Input Channel to Output Trigger Enable registers</td>
</tr>
<tr>
<td>External</td>
<td>CTIPIDR0</td>
<td>CTI Peripheral Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>CTIPIDR1</td>
<td>CTI Peripheral Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>CTIPIDR2</td>
<td>CTI Peripheral Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>CTIPIDR3</td>
<td>CTI Peripheral Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>CTIPIDR4</td>
<td>CTI Peripheral Identification Register 4</td>
</tr>
<tr>
<td>External</td>
<td>CTITRIGINSTATUS</td>
<td>CTI Trigger In Status register</td>
</tr>
<tr>
<td>External</td>
<td>CTITRIGOUTSTATUS</td>
<td>CTI Trigger Out Status register</td>
</tr>
</tbody>
</table>

In the Virt functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>ATS1HR</td>
<td>Address Translate Stage 1 Hyp mode Read</td>
</tr>
<tr>
<td>AArch32</td>
<td>ATS1HW</td>
<td>Address Translate Stage 1 Hyp mode Write</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHCTL</td>
<td>Counter-timer Hyp Control register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHP_CVAL</td>
<td>Counter-timer Hyp Physical CompareValue register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTHP_TVAL</td>
<td>Counter-timer Hyp Physical Timer TimerValue register</td>
</tr>
<tr>
<td>AArch32</td>
<td>CNTVOFF</td>
<td>Counter-timer Virtual Offset register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HACR</td>
<td>Hyp Auxiliary Configuration Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HACTLR</td>
<td>Hyp Auxiliary Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HACTLR2</td>
<td>Hyp Auxiliary Control Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>HADFSFR</td>
<td>Hyp Auxiliary Data Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HAIFSR</td>
<td>Hyp Auxiliary Instruction Fault Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HAMAIR0</td>
<td>Hyp Auxiliary Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>HAMAIR1</td>
<td>Hyp Auxiliary Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>HCPTR</td>
<td>Hyp Architectural Feature Trap Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HCR</td>
<td>Hyp Configuration Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HCR2</td>
<td>Hyp Configuration Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>HDR</td>
<td>Hyp Debug Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HDFAR</td>
<td>Hyp Data Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HIFAR</td>
<td>Hyp Instruction Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HMAIR0</td>
<td>Hyp Memory Attribute Indirection Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>HMAIR1</td>
<td>Hyp Memory Attribute Indirection Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>HPFAR</td>
<td>Hyp IPA Fault Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HRMR</td>
<td>Hyp Reset Management Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HSCTRLR</td>
<td>Hyp System Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HSR</td>
<td>Hyp Syndrome Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HSTR</td>
<td>Hyp System Trap Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HTCR</td>
<td>Hyp Translation Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HTPIDR</td>
<td>Hyp Software Thread ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HTRFCR</td>
<td>Hyp Trace Filter Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HTTBR</td>
<td>Hyp Translation Table Base Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>HVBAR</td>
<td>Hyp Vector Base Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HSRE</td>
<td>Interrupt Controller Hyp System Register Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_AP0R&lt;n&gt;</td>
<td>Interrupt Controller Hyp Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_AP1R&lt;n&gt;</td>
<td>Interrupt Controller Hyp Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_EISR</td>
<td>Interrupt Controller End of Interrupt Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_ELRSR</td>
<td>Interrupt Controller Empty List Register Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_HCR</td>
<td>Interrupt Controller Hyp Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_LR&lt;n&gt;</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_LRC&lt;n&gt;</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_MISR</td>
<td>Interrupt Controller Maintenance Interrupt State Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_VMCR</td>
<td>Interrupt Controller Virtual Machine Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICH_VTR</td>
<td>Interrupt Controller VGIC Type Register</td>
</tr>
<tr>
<td>Exec state</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIAALL</td>
<td>TLB Invalidate All, Hyp mode</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIAALLHIS</td>
<td>TLB Invalidate All, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIIPAS2</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIIPAS2IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIIPAS2L</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIIPAS2LIS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAH</td>
<td>TLB Invalidate by VA, Hyp mode</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAHIS</td>
<td>TLB Invalidate by VA, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVAL</td>
<td>TLB Invalidate by VA, Last level, Hyp mode</td>
</tr>
<tr>
<td>AArch32</td>
<td>TLBIMVALHIS</td>
<td>TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable</td>
</tr>
<tr>
<td>AArch32</td>
<td>VMPIDR</td>
<td>Virtualization Multiprocessor ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VFDR</td>
<td>Virtualization Processor ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VTCR</td>
<td>Virtualization Translation Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VTTBR</td>
<td>Virtualization Translation Table Base Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ACTLR_EL2</td>
<td>Auxiliary Control Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR0_EL2</td>
<td>Auxiliary Fault Status Register 0 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AFSR1_EL2</td>
<td>Auxiliary Fault Status Register 1 (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>AMAIR_EL2</td>
<td>Auxiliary Memory Attribute Indirection Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHCTL_EL2</td>
<td>Counter-timer Hypervisor Control register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHPS_CTL_EL2</td>
<td>Counter-timer Secure Physical Timer Control register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHPS_CVAL_EL2</td>
<td>Counter-timer Secure Physical Timer CompareValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHPS_TVAL_EL2</td>
<td>Counter-timer Secure Physical Timer TimeValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHP_CTL_EL2</td>
<td>Counter-timer Hypervisor Physical Timer Control register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHP_CVAL_EL2</td>
<td>Counter-timer Physical Timer CompareValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTHP_TVAL_EL2</td>
<td>Counter-timer Physical Timer TimeValue register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>CNTVOFF_EL2</td>
<td>Counter-timer Virtual Offset register</td>
</tr>
<tr>
<td>AArch64</td>
<td>CPTR_EL2</td>
<td>Architectural Feature Trap Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ESR_EL2</td>
<td>Exception Syndrome Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>FAR_EL2</td>
<td>Fault Address Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>HACR_EL2</td>
<td>Hypervisor Auxiliary Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>HCR_EL2</td>
<td>Hypervisor Configuration Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>HPFAR_EL2</td>
<td>Hypervisor IPA Fault Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>HSTR_EL2</td>
<td>Hypervisor System Trap Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_SRE_EL2</td>
<td>Interrupt Controller System Register Enable register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_AP0R&lt;n&gt;_EL2</td>
<td>Interrupt Controller Hyp Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH/APIR&lt;n&gt;_EL2</td>
<td>Interrupt Controller Hyp Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_EISR_EL2</td>
<td>Interrupt Controller End of Interrupt Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_ELSR_EL2</td>
<td>Interrupt Controller Empty List Register Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_HCR_EL2</td>
<td>Interrupt Controller Hyp Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_LR&lt;n&gt;_EL2</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_MISR_EL2</td>
<td>Interrupt Controller Maintenance Interrupt State Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_VMCR_EL2</td>
<td>Interrupt Controller Virtual Machine Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICH_VTR_EL2</td>
<td>Interrupt Controller VGIC Type Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>MAIR_EL2</td>
<td>Memory Attribute Indirection Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MDCR_EL2</td>
<td>Monitor Debug Configuration Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>RMR_EL2</td>
<td>Reset Management Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>SCTLR_EL2</td>
<td>System Control Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TCR_EL2</td>
<td>Translation Control Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1IS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1S</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1S</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIIPAS2E1OS</td>
<td>TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIRIPAS2E1</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIRIPAS2E1S</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable</td>
</tr>
<tr>
<td>AArch64</td>
<td>TLBIRIPAS2E1OS</td>
<td>TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable</td>
</tr>
</tbody>
</table>
### System Register index by functional group

#### Exec state | Name | Description
--- | --- | ---
AArch64 | TLBI RIPAS2LE1 | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1
AArch64 | TLBI RIPAS2LE1IS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable
AArch64 | TLBI RIPAS2LE1OS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable
AArch64 | TPIIDR_EL2 | EL2 Software Thread ID Register
AArch64 | TTBR0_EL2 | Translation Table Base Register 0 (EL2)
AArch64 | TTBR1_EL2 | Translation Table Base Register 1 (EL2)
AArch64 | VBAR_EL2 | Vector Base Address Register (EL2)
AArch64 | VMPIDR_EL2 | Virtualization Multiprocessor ID Register
AArch64 | VPIDR_EL2 | Virtualization Processor ID Register
AArch64 | VTCR_EL2 | Virtualization Translation Control Register
AArch64 | VTTR_EL2 | Virtualization Translation Table Base Register

**In the Secure functional group:**

#### Exec state | Name | Description
--- | --- | ---
AArch32 | ICC_MCTLR | Interrupt Controller Monitor Control Register
AArch32 | ICC_MSRE | Interrupt Controller Monitor System Register Enable register
AArch32 | MVBAR | Monitor Vector Base Address Register
AArch32 | NSACR | Non-Secure Access Control Register
AArch32 | SCR | Secure Configuration Register
AArch32 | SDR | Secure Debug Control Register
AArch32 | SDER | Secure Debug Enable Register
AArch64 | ACTLR_EL3 | Auxiliary Control Register (EL3)
AArch64 | AFSR0_EL3 | Auxiliary Fault Status Register 0 (EL3)
AArch64 | AFSR1_EL3 | Auxiliary Fault Status Register 1 (EL3)
AArch64 | AMAIR_EL3 | Auxiliary Memory Attribute Indirection Register (EL3)
AArch64 | CPTIR_EL3 | Architectural Feature Trap Register (EL3)
AArch64 | ICC_CTLR_EL3 | Interrupt Controller Control Register (EL3)
AArch64 | ICC_SRE_EL3 | Interrupt Controller System Register Enable register (EL3)
AArch64 | MDCR_EL3 | Monitor Debug Configuration Register (EL3)
AArch64 | SCR_EL3 | Secure Configuration Register
AArch64 | SDER32_EL3 | AArch32 Secure Debug Enable Register
AArch64 | VBAR_EL3 | Vector Base Address Register (EL3)

**In the Float functional group:**

#### Exec state | Name | Description
--- | --- | ---
AArch32 | FPEXC | Floating-Point Exception Control register
AArch32 | FPSCR | Floating-Point Status and Control Register
AArch32 | FPSID | Floating-Point System ID register
AArch32 | MVFR0 | Media and VFP Feature Register 0
AArch32 | MVFR1 | Media and VFP Feature Register 1
AArch32 | MVFR2 | Media and VFP Feature Register 2
AArch64 | FPCR | Floating-point Control Register
AArch64 | FPEXC32_EL2 | Floating-Point Exception Control register
AArch64 | FPSR | Floating-point Status Register
AArch64 | MVFR0_EL1 | AArch32 Media and VFP Feature Register 0
AArch64 | MVFR1_EL1 | AArch32 Media and VFP Feature Register 1
AArch64 | MVFR2_EL1 | AArch32 Media and VFP Feature Register 2

**In the Legacy functional group:**

#### Exec state | Name | Description
--- | --- | ---
AArch32 | CP15DMB | Data Memory Barrier System instruction
AArch32 | CP15DSB | Data Synchronization Barrier System instruction
AArch32 | CP15SIB | Instruction Synchronization Barrier System instruction
AArch32 | FCSEIDR | FCSE Process ID register
AArch32 | IDIR | Jazelle ID Register
### System Register index by functional group

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>JMCR</td>
<td>Jazelle Main Configuration Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>JOSCR</td>
<td>Jazelle OS Control Register</td>
</tr>
</tbody>
</table>

**In the Trace functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>TRCACLATR&lt;n&gt;</td>
<td>Address Comparator Access Type Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCACVR&lt;n&gt;</td>
<td>Address Comparator Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCAUTHSTATUS</td>
<td>Authentication Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCAUXCTLR</td>
<td>Auxiliary Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCBBCCTRL</td>
<td>Branch Broadcast Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCCCCCTRL</td>
<td>Cycle Count Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCICDCCCTRL0</td>
<td>Context Identifier Comparator Control Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCICDCCCTRL1</td>
<td>Context Identifier Comparator Control Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCICDVR&lt;n&gt;</td>
<td>Context Identifier Comparator Value Registers &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCCLAIMCLR</td>
<td>Claim Tag Clear Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCCLAIMSET</td>
<td>Claim Tag Set Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCCOUNTCTRL&lt;n&gt;</td>
<td>Counter Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCCNTLDR&lt;n&gt;</td>
<td>Counter Reload Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCCNTVR&lt;n&gt;</td>
<td>Counter Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCONFIGR</td>
<td>Trace Configuration Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCDEVARCH</td>
<td>Device Architecture Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCDEVID</td>
<td>Device Configuration Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCEVENTCTL0R</td>
<td>Event Control 0 Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCEVENTCTL1R</td>
<td>Event Control 1 Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCEXTINSEL&lt;n&gt;</td>
<td>External Input Select Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR0</td>
<td>ID Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR1</td>
<td>ID Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR10</td>
<td>ID Register 10</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR11</td>
<td>ID Register 11</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR12</td>
<td>ID Register 12</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR13</td>
<td>ID Register 13</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR2</td>
<td>ID Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR3</td>
<td>ID Register 3</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR4</td>
<td>ID Register 4</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR5</td>
<td>ID Register 5</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR6</td>
<td>ID Register 6</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR7</td>
<td>ID Register 7</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR8</td>
<td>ID Register 8</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIDR9</td>
<td>ID Register 9</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIMSPEC0</td>
<td>IMP DEF Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCIMSPEC&lt;n&gt;</td>
<td>IMP DEF Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCOSLSR</td>
<td>Trace OS Lock Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCPGRTLR</td>
<td>Programming Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCQCTRL</td>
<td>Q Element Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCRSTCTRL&lt;n&gt;</td>
<td>Resource Selection Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCRRS</td>
<td>Resources Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSEQEVR&lt;n&gt;</td>
<td>Sequencer State Transition Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSEQRSTEV&lt;R&gt;</td>
<td>Sequencer Reset Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSEQSTR</td>
<td>Sequencer State Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSSCCCR&lt;n&gt;</td>
<td>Single-shot Comparator Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSSCSR&lt;n&gt;</td>
<td>Single-shot Comparator Control Status Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSSPEC&lt;n&gt;</td>
<td>Single-shot Processing Element Comparator Input Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSTALLCTRL</td>
<td>Stall Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSTATR</td>
<td>Trace Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCSYMPR</td>
<td>Synchronization Period Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCTACTIDR</td>
<td>Trace ID Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCTSTCTRL</td>
<td>Timestamp Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCVICTLR</td>
<td>ViewInst Main Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCVIIECTRLR</td>
<td>ViewInst Include/Exclude Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCVIPCSSCTRL</td>
<td>ViewInst Start/Stop PE Comparator Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCVISSCTRL</td>
<td>ViewInst Start/Stop Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>TRCVMDCCTR0R</td>
<td>Virtual Context Identifier Comparator Control Register 0</td>
</tr>
</tbody>
</table>
### Exec state | Name | Description
--- | --- | ---
AArch64 | TRCVMIDCCTRLR1 | Virtual Context Identifier Comparator Control Register 1
AArch64 | TRCVMIDCVR<\(n\)> | Virtual Context Identifier Comparator Value Register \(<n\>)

---

**In the GIC functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>ICC_A0PR&lt;n&gt;</td>
<td>Interrupt Controller Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_A1PR&lt;n&gt;</td>
<td>Interrupt Controller Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_ASGI1R</td>
<td>Interrupt Controller Alias Software Generated Interrupt Group 1 Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_BPR0</td>
<td>Interrupt Controller Binary Point Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_BPR1</td>
<td>Interrupt Controller Binary Point Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_CTLR</td>
<td>Interrupt Controller Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_DIR</td>
<td>Interrupt Controller Deactivate Interrupt Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_EOIR0</td>
<td>Interrupt Controller End Of Interrupt Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_EOIR1</td>
<td>Interrupt Controller End Of Interrupt Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HPIR0</td>
<td>Interrupt Controller Highest Priority Pending Interrupt Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HPIR1</td>
<td>Interrupt Controller Highest Priority Pending Interrupt Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HSRE</td>
<td>Interrupt Controller Hyp System Register Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IAR0</td>
<td>Interrupt Controller Interrupt Acknowledge Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IAR1</td>
<td>Interrupt Controller Interrupt Acknowledge Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IGRPEN0</td>
<td>Interrupt Controller Interrupt Group 0 Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IGRPEN1</td>
<td>Interrupt Controller Interrupt Group 1 Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_MCTRL</td>
<td>Interrupt Controller Monitor Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_MGRPN1</td>
<td>Interrupt Controller Monitor Interrupt Group 1 Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_MSR</td>
<td>Interrupt Controller Monitor System Register Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_PM</td>
<td>Interrupt Controller Interrupt Priority Mask Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_ROM</td>
<td>Interrupt Controller Running Priority Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_SGI0R</td>
<td>Interrupt Controller Software Generated Interrupt Group 0 Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_SGI1R</td>
<td>Interrupt Controller Software Generated Interrupt Group 1 Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_SRE</td>
<td>Interrupt Controller System Register Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_A0PR&lt;n&gt;_EL1</td>
<td>Interrupt Controller Hyp Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_A1PR&lt;n&gt;_EL1</td>
<td>Interrupt Controller Hyp Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HYSP</td>
<td>Interrupt Controller End of Interrupt Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_EISR</td>
<td>Interrupt Controller Empty List Register Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_ELRSR</td>
<td>Interrupt Controller Empty List Register Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HCR</td>
<td>Interrupt Controller Hyp Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_LR&lt;n&gt;</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_LR&lt;n&gt;</td>
<td>Interrupt Controller List Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_MISR</td>
<td>Interrupt Controller Maintenance Interrupt State Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_VMCR</td>
<td>Interrupt Controller Virtual Machine Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_VTR</td>
<td>Interrupt Controller VGIC Type Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_A0PR&lt;n&gt;</td>
<td>Interrupt Controller Virtual Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_A1PR&lt;n&gt;</td>
<td>Interrupt Controller Virtual Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_BPR0</td>
<td>Interrupt Controller Virtual Binary Point Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_BPR1</td>
<td>Interrupt Controller Virtual Binary Point Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_CTLR</td>
<td>Interrupt Controller Virtual Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_DIR</td>
<td>Interrupt Controller Deactivate Virtual Interrupt Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_FOIR0</td>
<td>Interrupt Controller Virtual End Of Interrupt Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_FOIR1</td>
<td>Interrupt Controller Virtual End Of Interrupt Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HPPR0</td>
<td>Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_HPPR1</td>
<td>Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IAR0</td>
<td>Interrupt Controller Virtual Interrupt Acknowledge Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IAR1</td>
<td>Interrupt Controller Virtual Interrupt Acknowledge Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IGRPEN0</td>
<td>Interrupt Controller Virtual Interrupt Group 0 Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_IGRPEN1</td>
<td>Interrupt Controller Virtual Interrupt Group 1 Enable register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_PM</td>
<td>Interrupt Controller Virtual Interrupt Priority Mask Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ICC_BPR</td>
<td>Interrupt Controller Virtual Running Priority Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_A0PR&lt;n&gt;_EL1</td>
<td>Interrupt Controller Active Priorities Group 0 Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_A1PR&lt;n&gt;_EL1</td>
<td>Interrupt Controller Active Priorities Group 1 Registers</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_ASGI1R_EL1</td>
<td>Interrupt Controller Alias Software Generated Interrupt Group 1 Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_BPR0_EL1</td>
<td>Interrupt Controller Binary Point Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_BPR1_EL1</td>
<td>Interrupt Controller Binary Point Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_CTLR_EL1</td>
<td>Interrupt Controller Control Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>ICC_CTLR_EL3</td>
<td>Interrupt Controller Control Register (EL3)</td>
</tr>
</tbody>
</table>
### System Register index by functional group

**Exec state** | **Name** | **Description**
--- | --- | ---
AArch64 | ICC_DIR_EL1 | Interrupt Controller Deactivate Interrupt Register
AArch64 | ICC_EOFR0_EL1 | Interrupt Controller End Of Interrupt Register 0
AArch64 | ICC_EOFR1_EL1 | Interrupt Controller End Of Interrupt Register 1
AArch64 | ICC_HPPIR0_EL1 | Interrupt Controller Highest Priority Pending Interrupt Register 0
AArch64 | ICC_HPPIR1_EL1 | Interrupt Controller Highest Priority Pending Interrupt Register 1
AArch64 | ICC_IAR0_EL1 | Interrupt Controller Interrupt Acknowledge Register 0
AArch64 | ICC_IAR1_EL1 | Interrupt Controller Interrupt Acknowledge Register 1
AArch64 | ICC_IGRPEN0_EL1 | Interrupt Controller Interrupt Group 0 Enable register
AArch64 | ICC_IGRPEN1_EL1 | Interrupt Controller Interrupt Group 1 Enable register
AArch64 | ICC_IGRPEN1_EL3 | Interrupt Controller Interrupt Group 1 Enable register (EL3)
AArch64 | ICC_PMR_EL1 | Interrupt Controller Interrupt Priority Mask Register
AArch64 | ICC_RPR_EL1 | Interrupt Controller Running Priority Register
AArch64 | ICC_SGI0R_EL1 | Interrupt Controller Software Generated Interrupt Group 0 Register
AArch64 | ICC_SGI1R_EL1 | Interrupt Controller Software Generated Interrupt Group 1 Register
AArch64 | ICC_SRE_EL1 | Interrupt Controller System Register Enable register (EL1)
AArch64 | ICC_SRE_EL2 | Interrupt Controller System Register Enable register (EL2)
AArch64 | ICC_SRE_EL3 | Interrupt Controller System Register Enable register (EL3)
AArch64 | ICH_APO<n>_EL2 | Interrupt Controller Hyp Active Priorities Group 0 Registers
AArch64 | ICH_API<n>_EL2 | Interrupt Controller Hyp Active Priorities Group 1 Registers
AArch64 | ICH_ELRSR_EL2 | Interrupt Controller Empty List Register Status Register
AArch64 | ICH_HCR_EL2 | Interrupt Controller Hyp Control Register
AArch64 | ICH_LR<n>_EL2 | Interrupt Controller List Registers
AArch64 | ICH_MISR_EL2 | Interrupt Controller Maintenance Interrupt State Register
AArch64 | ICH_VMCR_EL2 | Interrupt Controller Virtual Machine Control Register
AArch64 | ICH_VTR_EL2 | Interrupt Controller VGIC Type Register
AArch64 | ICV_AP0R<n>_EL1 | Interrupt Controller Virtual Active Priorities Group 0 Registers
AArch64 | ICV_AP1R<n>_EL1 | Interrupt Controller Virtual Active Priorities Group 1 Registers
AArch64 | ICV_BPR0_EL1 | Interrupt Controller Virtual Binary Point Register 0
AArch64 | ICV_BPR1_EL1 | Interrupt Controller Virtual Binary Point Register 1
AArch64 | ICV_CTLR_EL1 | Interrupt Controller Virtual Control Register
AArch64 | ICV_DIR_EL1 | Interrupt Controller Deactivate Virtual Interrupt Register
AArch64 | ICV_EOFR0_EL1 | Interrupt Controller Virtual End Of Interrupt Register 0
AArch64 | ICV_EOFR1_EL1 | Interrupt Controller Virtual End Of Interrupt Register 1
AArch64 | ICV_HPPIR0_EL1 | Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0
AArch64 | ICV_HPPIR1_EL1 | Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1
AArch64 | ICV_IAR0_EL1 | Interrupt Controller Virtual Interrupt Acknowledge Register 0
AArch64 | ICV_IAR1_EL1 | Interrupt Controller Virtual Interrupt Acknowledge Register 1
AArch64 | ICV_IGRPEN0_EL1 | Interrupt Controller Virtual Interrupt Group 0 Enable register
AArch64 | ICV_IGRPEN1_EL1 | Interrupt Controller Virtual Interrupt Group 1 Enable register
AArch64 | ICV_PMR_EL1 | Interrupt Controller Virtual Interrupt Priority Mask Register
AArch64 | ICV_RPR_EL1 | Interrupt Controller Virtual Running Priority Register

**In the GICD functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GICD_CLRSP1_NSR</td>
<td>Clear Non-secure SPI Pending Register</td>
</tr>
<tr>
<td>External</td>
<td>GICD_CLRSP1_SR</td>
<td>Clear Secure SPI Pending Register</td>
</tr>
<tr>
<td>External</td>
<td>GICD_CPENDSGIR&lt;n&gt;</td>
<td>SGI Clear-Pending Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_CTLR</td>
<td>Distributor Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICACTIVER&lt;n&gt;</td>
<td>Interrupt Clear-Active Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICACTIVER&lt;n&gt;E</td>
<td>Interrupt Clear-Active Registers (extended SPI range)</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICENABLER&lt;n&gt;</td>
<td>Interrupt Clear-Enable Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICENABLER&lt;n&gt;E</td>
<td>Interrupt Clear-Enable Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICFGR&lt;n&gt;</td>
<td>Interrupt Configuration Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICFGR&lt;n&gt;E</td>
<td>Interrupt Configuration Registers (Extended SPI Range)</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICPENDR&lt;n&gt;</td>
<td>Interrupt Clear-Pending Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_ICPENDR&lt;n&gt;E</td>
<td>Interrupt Clear-Pending Registers (extended SPI range)</td>
</tr>
<tr>
<td>External</td>
<td>GICD_IGROUPR&lt;n&gt;</td>
<td>Interrupt Group Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_IGROUPR&lt;n&gt;E</td>
<td>Interrupt Group Registers (extended SPI range)</td>
</tr>
<tr>
<td>External</td>
<td>GICD_IGRPMODR&lt;n&gt;</td>
<td>Interrupt Group Modifier Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICD_IGRPMODR&lt;n&gt;E</td>
<td>Interrupt Group Modifier Registers (extended SPI range)</td>
</tr>
</tbody>
</table>
## System Register index by functional group

### Exec state | Name | Description
---|---|---
External | GICD_IDR | Distributor Implementer Identification Register
External | GICD_IPRIORITRY<n> | Interrupt Priority Registers
External | GICD_IPRIORITRY<n>E | Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.
External | GICD_IROUTER<n> | Interrupt Routing Registers
External | GICD_IROUTER<n>E | Interrupt Routing Registers (Extended SPI Range)
External | GICD_ISACTIVER<n> | Interrupt Set-Active Registers
External | GICD_ISACTIVER<n>E | Interrupt Set-Active Registers (extended SPI range)
External | GICD_ISENABLER<n> | Interrupt Set-Enable Registers
External | GICD_ISENABLER<n>E | Interrupt Set-Enable Registers
External | GICD_ISPENDR<n> | Interrupt Set-Pending Registers
External | GICD_ISPENDR<n>E | Interrupt Set-Pending Registers (extended SPI range)
External | GICD_ITARGETSR<n> | Interrupt Processor Targets Registers
External | GICD_NSACR<n> | Non-secure Access Control Registers
External | GICD_NSACR<n>E | Non-secure Access Control Registers
External | GICD_SETSPI_NS | Set Non-secure SPI Pending Register
External | GICD_SETSPI_SR | Set Secure SPI Pending Register
External | GICD_SGI | Software Generated Interrupt Register
External | GICD_SPENDSGIR<n> | SGI Set-Pending Registers
External | GICD_STATUSR | Error Reporting Status Register
External | GICD_TYPER | Interrupt Controller Type Register

### In the GICR functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GICR_CLRLPIR</td>
<td>Clear LPI Pending Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_CTLR</td>
<td>Redistributor Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICACTIVER0</td>
<td>Interrupt Clear-Active Register 0</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICACTIVER&lt;n&gt;</td>
<td>Interrupt Clear-Active Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICENABLER0</td>
<td>Interrupt Clear-Enable Register 0</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICENABLER&lt;n&gt;</td>
<td>Interrupt Clear-Enable Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICFGR0</td>
<td>Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICFGR1</td>
<td>Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICFGR&lt;n&gt;</td>
<td>Interrupt configuration registers</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICPENDR0</td>
<td>Interrupt Clear-Pending Register 0</td>
</tr>
<tr>
<td>External</td>
<td>GICR_ICPENDR&lt;n&gt;</td>
<td>Interrupt Clear-Pending Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICR_IGROUPR0</td>
<td>Interrupt Group Register 0</td>
</tr>
<tr>
<td>External</td>
<td>GICR_IGROUPR&lt;n&gt;</td>
<td>Interrupt Group Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICR_IGRPMODR0</td>
<td>Interrupt Group Modifier Register 0</td>
</tr>
<tr>
<td>External</td>
<td>GICR_IGRPMODR&lt;n&gt;</td>
<td>Interrupt Group Modifier Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICR_IDR</td>
<td>Redistributor Implementer Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_INVALLR</td>
<td>Redistributor Invalidate All Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_INVLPIR</td>
<td>Redistributor Invalidate LPI Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_IRPAMIDR</td>
<td>Report maximum PARTID and PMG Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_MPAMIDR</td>
<td>Report maximum PARTID and PMG Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_STATUSR</td>
<td>Error Reporting Status Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_PARTIDR</td>
<td>Set PARTID and PMG Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_PENDBASER</td>
<td>Redistributor LPI Pending Table Base Address Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_PROPPASER</td>
<td>Redistributor Properties Base Address Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_SETLPIR</td>
<td>Set LPI Pending Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_SYNR</td>
<td>Redistributor Synchronize Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_TYPER</td>
<td>Redistributor Type Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_VPENDBASER</td>
<td>Virtual Redistributor LPI Pending Table Base Address Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_VPROPPASER</td>
<td>Virtual Redistributor Properties Base Address Register</td>
</tr>
<tr>
<td>External</td>
<td>GICR_WAKE</td>
<td>Redistributor Wake Register</td>
</tr>
</tbody>
</table>
**In the GICC functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GICC_ABPR</td>
<td>CPU Interface Aliased Binary Point Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_AEOIR</td>
<td>CPU Interface Aliased End Of Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_AHPPIR</td>
<td>CPU Interface Aliased Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_AIAR</td>
<td>CPU Interface Aliased Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_APR&lt;n&gt;</td>
<td>CPU Interface Active Priorities Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICC_BPR</td>
<td>CPU Interface Binary Point Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_CTLR</td>
<td>CPU Interface Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_DIR</td>
<td>CPU Interface Deactivate Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_EOIR</td>
<td>CPU Interface End Of Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_HPPIR</td>
<td>CPU Interface Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_IAR</td>
<td>CPU Interface Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_IIDR</td>
<td>CPU Interface Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_NSAPR&lt;n&gt;</td>
<td>CPU Interface Non-secure Active Priorities Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICC_PMR</td>
<td>CPU Interface Priority Mask Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_RPR</td>
<td>CPU Interface Running Priority Register</td>
</tr>
<tr>
<td>External</td>
<td>GICC_STATUSR</td>
<td>CPU Interface Status Register</td>
</tr>
</tbody>
</table>

**In the GICV functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GICV_ABPR</td>
<td>Virtual Machine Aliased Binary Point Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_AEOIR</td>
<td>Virtual Machine Aliased End Of Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_AHPPIR</td>
<td>Virtual Machine Aliased Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_AIAR</td>
<td>Virtual Machine Aliased Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_APR&lt;n&gt;</td>
<td>Virtual Machine Active Priorities Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICV_BPR</td>
<td>Virtual Machine Binary Point Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_CTLR</td>
<td>Virtual Machine Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_DIR</td>
<td>Virtual Machine Deactivate Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_EOIR</td>
<td>Virtual Machine End Of Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_HPPIR</td>
<td>Virtual Machine Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_IAR</td>
<td>Virtual Machine Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_IIDR</td>
<td>Virtual Machine CPU Interface Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_PMR</td>
<td>Virtual Machine Priority Mask Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_RPR</td>
<td>Virtual Machine Running Priority Register</td>
</tr>
<tr>
<td>External</td>
<td>GICV_STATUSR</td>
<td>Virtual Machine Error Reporting Status Register</td>
</tr>
</tbody>
</table>

**In the GICH functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GICH_APR&lt;n&gt;</td>
<td>Active Priorities Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICH_EISR</td>
<td>End Interrupt Status Register</td>
</tr>
<tr>
<td>External</td>
<td>GICH_ELRSR</td>
<td>Empty List Register Status Register</td>
</tr>
<tr>
<td>External</td>
<td>GICH_HCR</td>
<td>Hypervisor Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GICH_LR&lt;n&gt;</td>
<td>List Registers</td>
</tr>
<tr>
<td>External</td>
<td>GICH_MISR</td>
<td>Maintenance Interrupt Status Register</td>
</tr>
<tr>
<td>External</td>
<td>GICH_VMCR</td>
<td>Virtual Machine Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GICH_VTR</td>
<td>Virtual Type Register</td>
</tr>
</tbody>
</table>

**In the GITS functional group:**

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GITS_BASER&lt;n&gt;</td>
<td>ITS Translation Table Descriptors</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CBASER</td>
<td>ITS Command Queue Descriptor</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CREADR</td>
<td>ITS Read Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CTLR</td>
<td>ITS Control Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CWRITER</td>
<td>ITS Write Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_IIDR</td>
<td>ITS Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_MPAMIDR</td>
<td>Report maximum PARTID and PMG Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_PARTIDR</td>
<td>Set PARTID and PMG Register</td>
</tr>
</tbody>
</table>
## System Register index by functional group

### External Registers

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GITS_TRANSLATER</td>
<td>ITS Translation Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_TYPER</td>
<td>ITS Type Register</td>
</tr>
</tbody>
</table>

### In the RAS functional group:

#### AArch32

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch32</td>
<td>DISR</td>
<td>Deferred Interrupt Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERRIDR</td>
<td>Error Record ID Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERSELR</td>
<td>Error Record Select Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXADDR</td>
<td>Selected Error Record Address Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXADDR2</td>
<td>Selected Error Record Address Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXCTRLR</td>
<td>Selected Error Record Control Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXCTRLR2</td>
<td>Selected Error Record Control Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXFR</td>
<td>Selected Error Record Feature Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXFR2</td>
<td>Selected Error Record Feature Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC0</td>
<td>Selected Error Record Miscellaneous Register 0</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC1</td>
<td>Selected Error Record Miscellaneous Register 1</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC2</td>
<td>Selected Error Record Miscellaneous Register 2</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC3</td>
<td>Selected Error Record Miscellaneous Register 3</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC4</td>
<td>Selected Error Record Miscellaneous Register 4</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC5</td>
<td>Selected Error Record Miscellaneous Register 5</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC6</td>
<td>Selected Error Record Miscellaneous Register 6</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXMMC7</td>
<td>Selected Error Record Miscellaneous Register 7</td>
</tr>
<tr>
<td>AArch32</td>
<td>ERXSTATUS</td>
<td>Selected Error Record Primary Status Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VDFSR</td>
<td>Virtual SError Exception Syndrome Register</td>
</tr>
<tr>
<td>AArch32</td>
<td>VDISR</td>
<td>Virtual Deferred Interrupt Status Register</td>
</tr>
</tbody>
</table>

#### AArch64

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>DISR_EL1</td>
<td>Deferred Interrupt Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERRIDR_EL1</td>
<td>Error Record ID Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERSELR_EL1</td>
<td>Error Record Select Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXADDR_EL1</td>
<td>Selected Error Record Address Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXCTRLR_EL1</td>
<td>Selected Error Record Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXFR_EL1</td>
<td>Selected Error Record Feature Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXMMC0_EL1</td>
<td>Selected Error Record Miscellaneous Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXMMC1_EL1</td>
<td>Selected Error Record Miscellaneous Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXMMC2_EL1</td>
<td>Selected Error Record Miscellaneous Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXMMC3_EL1</td>
<td>Selected Error Record Miscellaneous Register 3</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXPFGCDN_EL1</td>
<td>Selected Pseudo-fault Generation Countdown Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXPFGCTL_EL1</td>
<td>Selected Pseudo-fault Generation Control Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>ERXSTATUS_EL1</td>
<td>Selected Error Record Primary Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>VDISR_EL2</td>
<td>Virtual Deferred Interrupt Status Register</td>
</tr>
<tr>
<td>AArch64</td>
<td>VSESR_EL2</td>
<td>Virtual SError Exception Syndrome Register</td>
</tr>
</tbody>
</table>

#### External Registers

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;ADDR</td>
<td>Error Record Address Register</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;CTRLR</td>
<td>Error Record Control Register</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;FR</td>
<td>Error Record Feature Register</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;MISC0</td>
<td>Error Record Miscellaneous Register 0</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;MISC1</td>
<td>Error Record Miscellaneous Register 1</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;MISC2</td>
<td>Error Record Miscellaneous Register 2</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;MISC3</td>
<td>Error Record Miscellaneous Register 3</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;PFGCDN</td>
<td>Pseudo-fault Generation Countdown Register</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;PFGCTL</td>
<td>Pseudo-fault Generation Control Register</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;PFGF</td>
<td>Pseudo-fault Generation Feature Register</td>
</tr>
<tr>
<td>External</td>
<td>ERR&lt;n&gt;STATUS</td>
<td>Error Record Primary Status Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRCIDR0</td>
<td>Component Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>ERRCIDR1</td>
<td>Component Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>ERRCIDR2</td>
<td>Component Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>ERRCIDR3</td>
<td>Component Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>ERRCRICR0</td>
<td>Critical Error Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>External</td>
<td>ERRCRICR1</td>
<td>Critical Error Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>External</td>
<td>ERRCRICR2</td>
<td>Critical Error Interrupt Configuration Register 2</td>
</tr>
<tr>
<td>External</td>
<td>ERRDEVAFF</td>
<td>Device Affinity Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRDEVARCH</td>
<td>Device Architecture Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRDEVCONFIG</td>
<td>Device Configuration Register</td>
</tr>
</tbody>
</table>
### System Register index by functional group

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>ERRERICR0</td>
<td>Error Recovery Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>External</td>
<td>ERRERICR1</td>
<td>Error Recovery Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>External</td>
<td>ERRERICR2</td>
<td>Error Recovery Interrupt Configuration Register 2</td>
</tr>
<tr>
<td>External</td>
<td>ERRFHICR0</td>
<td>Fault-Handling Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>External</td>
<td>ERRFHICR1</td>
<td>Fault-Handling Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>External</td>
<td>ERRFHICR2</td>
<td>Fault-Handling Interrupt Configuration Register 2</td>
</tr>
<tr>
<td>External</td>
<td>ERRGSR</td>
<td>Error Group Status Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRIDR</td>
<td>Implementation Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRROCR&lt;n&gt;</td>
<td>Generic Error Interrupt Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRROS R</td>
<td>Error Interrupt Status Register</td>
</tr>
<tr>
<td>External</td>
<td>ERRPIDR0</td>
<td>Peripheral Identification Register 0</td>
</tr>
<tr>
<td>External</td>
<td>ERRPIDR1</td>
<td>Peripheral Identification Register 1</td>
</tr>
<tr>
<td>External</td>
<td>ERRPIDR2</td>
<td>Peripheral Identification Register 2</td>
</tr>
<tr>
<td>External</td>
<td>ERRPIDR3</td>
<td>Peripheral Identification Register 3</td>
</tr>
<tr>
<td>External</td>
<td>ERRPIDR4</td>
<td>Peripheral Identification Register 4</td>
</tr>
</tbody>
</table>

### In the MPAM functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>MPAM0_EL1</td>
<td>MPAM0 Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAM1_EL1</td>
<td>MPAM1 Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAM2_EL2</td>
<td>MPAM2 Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAM3_EL3</td>
<td>MPAM3 Register (EL3)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMHCR_EL2</td>
<td>MPAM Hypervisor Control Register (EL2)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMIDR_EL1</td>
<td>MPAM ID Register (EL1)</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM0_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 0</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM1_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 1</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM2_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 2</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM3_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 3</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM4_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 4</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM5_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 5</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM6_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 6</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVM7_EL2</td>
<td>MPAM Virtual PARTID Mapping Register 7</td>
</tr>
<tr>
<td>AArch64</td>
<td>MPAMVPVMV_EL2</td>
<td>MPAM Virtual Partition Mapping Valid Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_CMAX</td>
<td>MPAM Cache Maximum Capacity Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_CBPM</td>
<td>MPAM Cache Portion Bitmap Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_INTPARTID</td>
<td>MPAM Internal PARTID Narrowing Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_MBW_MAX</td>
<td>MPAM Memory Bandwidth Maximum Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_MBW_MIN</td>
<td>MPAM Cache Maximum Capacity Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_MBW_PBM</td>
<td>MPAM Bandwidth Portion Bitmap Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_MBW_PROP</td>
<td>MPAM Memory Bandwidth Proportional Stride Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_MBW_WINWD</td>
<td>MPAM Memory Bandwidth Partitioning Window Width Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_PART_SEL</td>
<td>MPAM Partition Configuration Selection Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMCFG_PRI</td>
<td>MPAM Priority Partition Configuration Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_AIDR</td>
<td>MPAM Architecture Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_CCAP_IDR</td>
<td>MPAM Features Cache Capacity Partitioning ID register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_CPOR_IDR</td>
<td>MPAM Features Cache Portion Partitioning ID register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_CSSMON_IDR</td>
<td>MPAM Features Cache Storage Usage Monitoring ID register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_ECR</td>
<td>MPAM Error Control Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_ESR</td>
<td>MPAM Error Status Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_IDR</td>
<td>MPAM Features Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_IIDR</td>
<td>MPAM Implementation Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_IMPL_IDR</td>
<td>MPAM Implementation-Specific Partitioning Feature Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_MBWUMON_IDR</td>
<td>MPAM Features Memory Bandwidth Usage Monitoring ID register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_MBW_IDR</td>
<td>MPAM Memory Bandwidth Partitioning Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_MSMON_IDR</td>
<td>MPAM Resource Monitoring Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_PARTID_NRW_IDR</td>
<td>MPAM PARTID Narrowing ID register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_PRI_IDR</td>
<td>MPAM Priority Partitioning Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>MPAMF_SIDR</td>
<td>MPAM Features Secure Identification Register</td>
</tr>
</tbody>
</table>
### System Register index by functional group

#### In the Pointer authentication functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AArch64</td>
<td>APDAKeyHi_EL1</td>
<td>Pointer Authentication Key A for Data (bits[127:64])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APDAKeyLo_EL1</td>
<td>Pointer Authentication Key A for Data (bits[63:0])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APDBKeyHi_EL1</td>
<td>Pointer Authentication Key B for Data (bits[127:64])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APDBKeyLo_EL1</td>
<td>Pointer Authentication Key B for Data (bits[63:0])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APGAKeyHi_EL1</td>
<td>Pointer Authentication Key A for Code (bits[127:64])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APGAKeyLo_EL1</td>
<td>Pointer Authentication Key A for Code (bits[63:0])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APIAKeyHi_EL1</td>
<td>Pointer Authentication Key A for Instruction (bits[127:64])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APIAKeyLo_EL1</td>
<td>Pointer Authentication Key A for Instruction (bits[63:0])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APIBKeyHi_EL1</td>
<td>Pointer Authentication Key B for Instruction (bits[127:64])</td>
</tr>
<tr>
<td>AArch64</td>
<td>APIBKeyLo_EL1</td>
<td>Pointer Authentication Key B for Instruction (bits[63:0])</td>
</tr>
</tbody>
</table>

#### In the GIC ITS registers functional group:

<table>
<thead>
<tr>
<th>Exec state</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External</td>
<td>GITS_BASER&lt;n&gt;</td>
<td>ITS Translation Table Descriptors</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CBASER</td>
<td>ITS Command Queue Descriptor</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CREADR</td>
<td>ITS Read Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CTLR</td>
<td>ITS Control Queue Descriptor</td>
</tr>
<tr>
<td>External</td>
<td>GITS_CWRITER</td>
<td>ITS Write Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_IIDR</td>
<td>ITS Identification Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_MPAMIDR</td>
<td>Report maximum PARTID and PMG Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_PARTIDR</td>
<td>Set PARTID and PMG Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_TRANSLATER</td>
<td>ITS Translation Register</td>
</tr>
<tr>
<td>External</td>
<td>GITS_TYPER</td>
<td>ITS Type Register</td>
</tr>
</tbody>
</table>

---

27/09/2019 18:23

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
External System registers

**AMCFGR**: Activity Monitors Configuration Register

**AMCGCR**: Activity Monitors Counter Group Configuration Register

**AMCIDR0**: Activity Monitors Component Identification Register 0

**AMCIDR1**: Activity Monitors Component Identification Register 1

**AMCIDR2**: Activity Monitors Component Identification Register 2

**AMCIDR3**: Activity Monitors Component Identification Register 3

**AMCNTENCLR0**: Activity Monitors Count Enable Clear Register 0

**AMCNTENCLR1**: Activity Monitors Count Enable Clear Register 1

**AMCNTENSET0**: Activity Monitors Count Enable Set Register 0

**AMCNTENSET1**: Activity Monitors Count Enable Set Register 1

**AMCR**: Activity Monitors Control Register

**AMDEVAFF0**: Activity Monitors Device Affinity Register 0

**AMDEVAFF1**: Activity Monitors Device Affinity Register 1

**AMDEVARCH**: Activity Monitors Device Architecture Register

**AMDEVTYPE**: Activity Monitors Device Type Register

**AMEVCNTR0<n>**: Activity Monitors Event Counter Registers 0

**AMEVCNTR1<n>**: Activity Monitors Event Counter Registers 1

**AMEVTYPER0<n>**: Activity Monitors Event Type Registers 0

**AMEVTYPER1<n>**: Activity Monitors Event Type Registers 1

**AIMIDR**: Activity Monitors Implementation Identification Register

**AMPIDR0**: Activity Monitors Peripheral Identification Register 0

**AMPIDR1**: Activity Monitors Peripheral Identification Register 1

**AMPIDR2**: Activity Monitors Peripheral Identification Register 2

**AMPIDR3**: Activity Monitors Peripheral Identification Register 3

**AMPIDR4**: Activity Monitors Peripheral Identification Register 4

**ASICCTL**: CTI External Multiplexer Control register

**CNTACR<n>**: Counter-timer Access Control Registers

**CNTCR**: Counter Control Register

**CNTCV**: Counter Count Value register

**CNTEL0ACR**: Counter-timer EL0 Access Control Register

**CNTFID0**: Counter Frequency ID

**CNTFID<n>**: Counter Frequency IDs, n > 0

**CNTFRQ**: Counter-timer Frequency

**CNTID**: Counter Identification Register
CNTNSAR: Counter-timer Non-secure Access Register
CNPCT: Counter-timer Physical Count
CTP_CTL: Counter-timer Physical Timer Control
CTP_CVAL: Counter-timer Physical Timer CompareValue
CTP_TVAL: Counter-timer Physical Timer TimerValue
CNTSCR: Counter Scale Register
CNTSR: Counter Status Register
CNTTIDR: Counter-timer Timer ID Register
CNTVCT: Counter-timer Virtual Count
CNTVOFF: Counter-timer Virtual Offset
CNTVOFF<n>: Counter-timer Virtual Offsets
CNTV_CTL: Counter-timer Virtual Timer Control
CNTV_CVAL: Counter-timer Virtual Timer CompareValue
CNTV_TVAL: Counter-timer Virtual Timer TimerValue
CTIAPPCLEAR: CTI Application Trigger Clear register
CTIAPPULSE: CTI Application Pulse register
CTIAPPSET: CTI Application Trigger Set register
CTIAUTHSTATUS: CTI Authentication Status register
CTICHINSTATUS: CTI Channel In Status register
CTICHOUTSTATUS: CTI Channel Out Status register
CTICIDR0: CTI Component Identification Register 0
CTICIDR1: CTI Component Identification Register 1
CTICIDR2: CTI Component Identification Register 2
CTICIDR3: CTI Component Identification Register 3
CTICLAIMCLR: CTI CLAIM Tag Clear register
CTICLAIMSET: CTI CLAIM Tag Set register
CTICONTROL: CTI Control register
CTIDEVAFF0: CTI Device Affinity register 0
CTIDEVAFF1: CTI Device Affinity register 1
CTIDEVAFF: CTI Device Architecture register
CTIDEVCTL: CTI Device Control register
CTIDEVID: CTI Device ID register 0
CTIDEVID1: CTI Device ID register 1
CTIDEVID2: CTI Device ID register 2
CTIDEVTYPE: CTI Device Type register
CTIGATE: CTI Channel Gate Enable register
CTIINEN<n>: CTI Input Trigger to Output Channel Enable registers
CTIINTACK: CTI Output Trigger Acknowledge register
CTIITCTRL: CTI Integration mode Control register
CTILAR: CTI Lock Access Register
CTILSR: CTI Lock Status Register
CTIOUTEN<n>: CTI Input Channel to Output Trigger Enable registers
CTIPIDR0: CTI Peripheral Identification Register 0
CTIPIDR1: CTI Peripheral Identification Register 1
CTIPIDR2: CTI Peripheral Identification Register 2
CTIPIDR3: CTI Peripheral Identification Register 3
CTIPIDR4: CTI Peripheral Identification Register 4
CTITRIGINSTATUS: CTI Trigger In Status register
CTITRIGOUTSTATUS: CTI Trigger Out Status register
CounterID<n>: Counter ID registers
DBGAUTHSTATUS_EL1: Debug Authentication Status register
DBGBCR<n>_EL1: Debug Breakpoint Control Registers
DBGBCR<n>_EL1: Debug Breakpoint Value Registers
DBGCLAIMCLR_EL1: Debug CLAIM Tag Clear register
DBGCLAIMSET_EL1: Debug CLAIM Tag Set register
DBGDTRRX_EL0: Debug Data Transfer Register, Receive
DBGDTRTX_EL0: Debug Data Transfer Register, Transmit
DBGWCR<n>_EL1: Debug Watchpoint Control Registers
DBGWVR<n>_EL1: Debug Watchpoint Value Registers
EDAA32PFR: External Debug AArch32 Processor Feature Register
EDACR: External Debug Auxiliary Control Register
EDCIDR0: External Debug Component Identification Register 0
EDCIDR1: External Debug Component Identification Register 1
EDCIDR2: External Debug Component Identification Register 2
EDCIDR3: External Debug Component Identification Register 3
EDCISRSR: External Debug Context ID Sample Register
EDDEVAFF0: External Debug Device Affinity register 0
EDDEVAFF1: External Debug Device Affinity register 1
EDDEVARCH: External Debug Device Architecture register
EDDEVVID: External Debug Device ID register 0
EDDEVVID1: External Debug Device ID register 1
EDDEVVID2: External Debug Device ID register 2
EDDEVTYPE: External Debug Device Type register
EDDFR: External Debug Feature Register
EDECCR: External Debug Exception Catch Control Register
EDECR: External Debug Execution Control Register
EDESR: External Debug Event Status Register
EDITCTRL: External Debug Integration mode Control register
EDITIR: External Debug Instruction Transfer Register
EDLAR: External Debug Lock Access Register
EDLSR: External Debug Lock Status Register
EDPCSR: External Debug Program Counter Sample Register
EDPIDR0: External Debug Peripheral Identification Register 0
EDPIDR1: External Debug Peripheral Identification Register 1
EDPIDR2: External Debug Peripheral Identification Register 2
EDPIDR3: External Debug Peripheral Identification Register 3
EDPIDR4: External Debug Peripheral Identification Register 4
EDPRCR: External Debug Power/Reset Control Register
EDPRSR: External Debug Processor Status Register
EDRCR: External Debug Reserve Control Register
EDSCR: External Debug Status and Control Register
EDVIDSR: External Debug Virtual Context Sample Register
EDWAR: External Debug Watchpoint Address Register
ERR<n>ADDR: Error Record Address Register
ERR<n>CTLR: Error Record Control Register
ERR<n>FR: Error Record Feature Register
ERR<n>MISC0: Error Record Miscellaneous Register 0
ERR<n>MISC1: Error Record Miscellaneous Register 1
ERR<n>MISC2: Error Record Miscellaneous Register 2
ERR<n>MISC3: Error Record Miscellaneous Register 3
ERR<n>PFGCDN: Pseudo-fault Generation Countdown Register
ERR<n>PFGCTL: Pseudo-fault Generation Control Register
ERR<n>PFGF: Pseudo-fault Generation Feature Register
ERR<n>STATUS: Error Record Primary Status Register
ERRCIDR0: Component Identification Register 0
ERRCIDR1: Component Identification Register 1
ERRCIDR2: Component Identification Register 2
ERRCIDR3: Component Identification Register 3
ERRCRICR0: Critical Error Interrupt Configuration Register 0
ERRCRICR1: Critical Error Interrupt Configuration Register 1
ERRCRICR2: Critical Error Interrupt Configuration Register 2
ERRDEVAFF: Device Affinity Register
ERRDEVARCH: Device Architecture Register
ERRDEVID: Device Configuration Register
ERRERICR0: Error Recovery Interrupt Configuration Register 0
ERRERICR1: Error Recovery Interrupt Configuration Register 1
ERRERICR2: Error Recovery Interrupt Configuration Register 2
ERRFHICR0: Fault-Handling Interrupt Configuration Register 0
ERRFHICR1: Fault-Handling Interrupt Configuration Register 1
ERRFHICR2: Fault-Handling Interrupt Configuration Register 2
ERRGSR: Error Group Status Register
ERRIIDR: Implementation Identification Register
ERRIROCR<n>: Generic Error Interrupt Configuration Register
ERRIRQSR: Error Interrupt Status Register
ERRPIDR0: Peripheral Identification Register 0
ERRPIDR1: Peripheral Identification Register 1
ERRPIDR2: Peripheral Identification Register 2
ERRPIDR3: Peripheral Identification Register 3
ERRPIDR4: Peripheral Identification Register 4
GICC_ABPR: CPU Interface Aliased Binary Point Register
GICC_AEOIR: CPU Interface Aliased End Of Interrupt Register
GICC_AHPPIR: CPU Interface Aliased Highest Priority Pending Interrupt Register
GICC_AIAR: CPU Interface Aliased Interrupt Acknowledge Register
GICC_APR<n>: CPU Interface Active Priorities Registers
GICC_BPR: CPU Interface Binary Point Register
GICC_CTLR: CPU Interface Control Register
GICC_DIR: CPU Interface Deactivate Interrupt Register
GICC_EQIR: CPU Interface End Of Interrupt Register
GICC_HPPIR: CPU Interface Highest Priority Pending Interrupt Register
GICC_JAR: CPU Interface Interrupt Acknowledge Register
GICC_IIDR: CPU Interface Identification Register
GICC_NSAPR<n>: CPU Interface Non-secure Active Priorities Registers
GICC_PMR: CPU Interface Priority Mask Register
GICC_RPR: CPU Interface Running Priority Register
GICC_STATUSR: CPU Interface Status Register
GICD_CLRSPI_NSR: Clear Non-secure SPI Pending Register
GICD_CLRSPI_SR: Clear Secure SPI Pending Register
GICD_CPENDSGIR<n>: SGI Clear-Pending Registers
GICD_CTLR: Distributor Control Register
GICD_ICACTIVER<n>: Interrupt Clear-Active Registers
GICD_ICACTIVER<n>E: Interrupt Clear-Active Registers (extended SPI range)
GICD_ICENABLER<n>: Interrupt Clear-Enable Registers
GICD_ICENABLER<n>E: Interrupt Clear-Enable Registers (extended SPI range)
GICD_ICFGR<n>: Interrupt Configuration Registers
GICD_ICFGR<n>E: Interrupt Configuration Registers (Extended SPI Range)
GICD_ICPENDR<n>: Interrupt Clear-Pending Registers
GICD_ICPENDR<n>E: Interrupt Clear-Pending Registers (extended SPI range)
GICD_IGROUPR<n>: Interrupt Group Registers
GICD_IGROUPR<n>E: Interrupt Group Registers (extended SPI range)
GICD_IGRPMODR<n>: Interrupt Group Modifier Registers
GICD_IGRPMODR<n>E: Interrupt Group Modifier Registers (extended SPI range)
GICD_IIDR: Distributor Implementer Identification Register
GICD_IPRIORITYR<n>: Interrupt Priority Registers
GICD_IPRIORITYR<n>E: Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.
GICD_IROUTER<n>: Interrupt Routing Registers
GICD_IROUTER<n>E: Interrupt Routing Registers (Extended SPI Range)
GICD_ISACTIVER<n>: Interrupt Set-Active Registers
GICD_ISACTIVER<n>E: Interrupt Set-Active Registers (extended SPI range)
GICD_ISENABLER<n>: Interrupt Set-Enable Registers
GICD_ISENABLER<n>E: Interrupt Set-Enable Registers (extended SPI range)
GICD_ISPENDR<n>: Interrupt Set-Pending Registers
GICD_ISPENDR<n>E: Interrupt Set-Pending Registers (extended SPI range)
GICD_ITARGETSR<n>: Interrupt Processor Targets Registers
GICD_NSACR<n>: Non-secure Access Control Registers
GICD_NSACR<n>E: Non-secure Access Control Registers
GICD_SETSPI_NSR: Set Non-secure SPI Pending Register
GICD_SETSPI_SR: Set Secure SPI Pending Register
GICD_SGIR: Software Generated Interrupt Register
GICD_SPENDSGIR<n>: SGI Set-Pending Registers
**GICD_STATUSR**: Error Reporting Status Register

**GICD_TYPER**: Interrupt Controller Type Register

**GICH_APR<n>**: Active Priorities Registers

**GICH_EISR**: End Interrupt Status Register

**GICH_ELRSR**: Empty List Register Status Register

**GICH_HCR**: Hypervisor Control Register

**GICH_LR<n>**: List Registers

**GICH_MISR**: Maintenance Interrupt Status Register

**GICH_VMCR**: Virtual Machine Control Register

**GICH_VTR**: Virtual Type Register

**GICR_CLRLPIR**: Clear LPI Pending Register

**GICR_CTLR**: Redistributor Control Register

**GICR_ICACTIVER0**: Interrupt Clear-Active Register 0

**GICR_ICACTIVER<n>E**: Interrupt Clear-Active Registers

**GICR_ICENABLER0**: Interrupt Clear-Enable Register 0

**GICR_ICENABLER<n>E**: Interrupt Clear-Enable Registers

**GICR_ICFGR0**: Interrupt Configuration Register 0

**GICR_ICFGR1**: Interrupt Configuration Register 1

**GICR_ICFGR<n>E**: Interrupt configuration registers

**GICR_ICPENDR0**: Interrupt Clear-Pending Register 0

**GICR_ICPENDR<n>E**: Interrupt Clear-Pending Registers

**GICR_IGROUPR0**: Interrupt Group Register 0

**GICR_IGROUPR<n>E**: Interrupt Group Registers

**GICR_IGRPMODR0**: Interrupt Group Modifier Register 0

**GICR_IGRPMODR<n>E**: Interrupt Group Modifier Registers

**GICR_IIDR**: Redistributor Implementer Identification Register

**GICR_INVALLR**: Redistributor Invalidate All Register

**GICR_INVLPISR**: Redistributor Invalidate LPI Register

**GICR_IPRIORITY<n>**: Interrupt Priority Registers

**GICR_IPRIORITY<n>E**: Interrupt Priority Registers (extended PPI range)

**GICR_ISACTIVER0**: Interrupt Set-Active Register 0

**GICR_ISACTIVER<n>E**: Interrupt Set-Active Registers

**GICR_ISENABLER0**: Interrupt Set-Enable Register 0

**GICR_ISENABLER<n>E**: Interrupt Set-Enable Registers

**GICR_ISPENDR0**: Interrupt Set-Pending Register 0

**GICR_ISPENDR<n>E**: Interrupt Set-Pending Registers
External System registers

GICR_MPAMIDR: Report maximum PARTID and PMG Register
GICR_NSACR: Non-secure Access Control Register
GICR_PARTIDR: Set PARTID and PMG Register
GICR_PENDBASER: Redistributor LPI Pending Table Base Address Register
GICR_PROPBASER: Redistributor Properties Base Address Register
GICR_SETLPISR: Set LPI Pending Register
GICR_STATUSR: Error Reporting Status Register
GICR_SYNCR: Redistributor Synchronize Register
GICR_TYPER: Redistributor Type Register
GICR_VPENDBASER: Virtual Redistributor LPI Pending Table Base Address Register
GICR_VPROPBASER: Virtual Redistributor Properties Base Address Register
GICR_WAKER: Redistributor Wake Register
GICV_ABPR: Virtual Machine Aliased Binary Point Register
GICV_AEOIR: Virtual Machine Aliased End Of Interrupt Register
GICV_AHPPIR: Virtual Machine Aliased Highest Priority Pending Interrupt Register
GICV_AIAR: Virtual Machine Aliased Interrupt Acknowledge Register
GICV_APR<n>: Virtual Machine Active Priorities Registers
GICV_BPR: Virtual Machine Binary Point Register
GICV_CTLR: Virtual Machine Control Register
GICV_DIR: Virtual Machine Deactivate Interrupt Register
GICV_EOIR: Virtual Machine End Of Interrupt Register
GICV_HPPIR: Virtual Machine Highest Priority Pending Interrupt Register
GICV_JAR: Virtual Machine Interrupt Acknowledge Register
GICV_JIDR: Virtual Machine CPU Interface Identification Register
GICV_PMR: Virtual Machine Priority Mask Register
GICV_RPR: Virtual Machine Running Priority Register
GICV_STATUSR: Virtual Machine Error Reporting Status Register
GITS_BASER<n>: ITS Translation Table Descriptors
GITS_CBASER: ITS Command Queue Descriptor
GITS_CREADR: ITS Read Register
GITS_CTLR: ITS Control Register
GITS_CWRITER: ITS Write Register
GITS_IIDR: ITS Identification Register
GITS_MPAMIDR: Report maximum PARTID and PMG Register
GITS_PARTIDR: Set PARTID and PMG Register
GITS_TRANSLATER: ITS Translation Register
GITS_TYPER: ITS Type Register
MIDR_EL1: Main ID Register
MPAMCFG_CMAX: MPAM Cache Maximum Capacity Partition Configuration Register
MPAMCFG_CBPM: MPAM Cache Portion Bitmap Partition Configuration Register
MPAMCFG_INTPARTID: MPAM Internal PARTID Narrowing Configuration Register
MPAMCFG_MBW_MAX: MPAM Memory Bandwidth Maximum Partition Configuration Register
MPAMCFG_MBW_MIN: MPAM Cache Maximum Capacity Partition Configuration Register
MPAMCFG_MBW_PBM: MPAM Bandwidth Portion Bitmap Partition Configuration Register
MPAMCFG_MBW_PROP: MPAM Memory Bandwidth Proportional Stride Partition Configuration Register
MPAMCFG_MBW_WINWD: MPAM Memory Bandwidth Partitioning Window Width Configuration Register
MPAMCFG_PART_SEL: MPAM Partition Configuration Selection Register
MPAMCFG_PRI: MPAM Priority Partition Configuration Register
MPAMF_AIDR: MPAM Architecture Identification Register
MPAMF_CCAP_IDR: MPAM Features Cache Capacity Partitioning ID register
MPAMF_CPOR_IDR: MPAM Features Cache Portion Partitioning ID register
MPAMF_CSUMON_IDR: MPAM Features Cache Storage Usage Monitoring ID register
MPAMF_ECR: MPAM Error Control Register
MPAMF_ESR: MPAM Error Status Register
MPAMF_IDR: MPAM Features Identification Register
MPAMF_IIDR: MPAM Implementation Identification Register
MPAMF_IMPL_IDR: MPAM Implementation-Specific Partitioning Feature Identification Register
MPAMF_MBWUMON_IDR: MPAM Features Memory Bandwidth Usage Monitoring ID register
MPAMF_MBW_IDR: MPAM Memory Bandwidth Partitioning Identification Register
MPAMF_MSMON_IDR: MPAM Resource Monitoring Identification Register
MPAMF_PARTID_NRW_IDR: MPAM PARTID Narrowing ID register
MPAMF_PRI_IDR: MPAM Priority Partitioning Identification Register
MPAMF_SIDR: MPAM Features Secure Identification Register
MSMON_CAPT_EVNT: MPAM Capture Event Generation Register
MSMON_CFG_CSU_CTL: MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register
MSMON_CFG_CSU_FLT: MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register
MSMON_CFG_MBWU_CTL: MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register
MSMON_CFG_MBWU_FLT: MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register
MSMON_CFG_MON_SEL: MPAM Monitor Instance Selection Register
MSMON_CSU: MPAM Cache Storage Usage Monitor Register
MSMON_CSU_CAPTURE: MPAM Cache Storage Usage Monitor Capture Register
EXTERNAL SYSTEM REGISTERS

- **MSMON_MBWU**: MPAM Memory Bandwidth Usage Monitor Register
- **MSMON_MBWU_CAPTURE**: MPAM Memory Bandwidth Usage Monitor Capture Register
- **MSMON_MBWU_L**: MPAM Long Memory Bandwidth Usage Monitor Register
- **MSMON_MBWU_L_CAPTURE**: MPAM Long Memory Bandwidth Usage Monitor Capture Register
- **OSLAR_EL1**: OS Lock Access Register
- **PMAUTHSTATUS**: Performance Monitors Authentication Status register
- **PMCCFILTR_EL0**: Performance Monitors Cycle Counter Filter Register
- **PMCCNTR_EL0**: Performance Monitors Cycle Counter
- **PMCEID0**: Performance Monitors Common Event Identification register 0
- **PMCEID1**: Performance Monitors Common Event Identification register 1
- **PMCEID2**: Performance Monitors Common Event Identification register 2
- **PMCEID3**: Performance Monitors Common Event Identification register 3
- **PMCFGR**: Performance Monitors Configuration Register
- **PMCID1SR**: CONTEXTIDR_EL1 Sample Register
- **PMCID2SR**: CONTEXTIDR_EL2 Sample Register
- **PMCIDR0**: Performance Monitors Component Identification Register 0
- **PMCIDR1**: Performance Monitors Component Identification Register 1
- **PMCIDR2**: Performance Monitors Component Identification Register 2
- **PMCIDR3**: Performance Monitors Component Identification Register 3
- **PMCNTENCLR_EL0**: Performance Monitors Count Enable Clear register
- **PMCNTENSET_EL0**: Performance Monitors Count Enable Set register
- **PMCR_EL0**: Performance Monitors Control Register
- **PMDEVAFF0**: Performance Monitors Device Affinity register 0
- **PMDEVAFF1**: Performance Monitors Device Affinity register 1
- **PMDEVARCH**: Performance Monitors Device Architecture register
- **PMDEVID**: Performance Monitors Device ID register
- **PMDEVTYPE**: Performance Monitors Device Type register
- **PMVNCNT<1)_EL0**: Performance Monitors Event Count Registers
- **PMVTYPE<1)_EL0**: Performance Monitors Event Type Registers
- **PMINTENCLR_EL1**: Performance Monitors Interrupt Enable Clear register
- **PMINTENSET_EL1**: Performance Monitors Interrupt Enable Set register
- **PMITCTRL**: Performance Monitors Integration mode Control register
- **PMLAR**: Performance Monitors Lock Access Register
- **PMLSR**: Performance Monitors Lock Status Register
- **PMMIR**: Performance Monitors Machine Identification Register
- **PMOVSCLR_EL0**: Performance Monitors Overflow Flag Status Clear register
PMOVSET_EL0: Performance Monitors Overflow Flag Status Set register

PMPCSR: Program Counter Sample Register

PMPIDR0: Performance Monitors Peripheral Identification Register 0

PMPIDR1: Performance Monitors Peripheral Identification Register 1

PMPIDR2: Performance Monitors Peripheral Identification Register 2

PMPIDR3: Performance Monitors Peripheral Identification Register 3

PMPIDR4: Performance Monitors Peripheral Identification Register 4

PMSWINC_EL0: Performance Monitors Software Increment register

PMVIDSR: VMID Sample Register

TRCACATR<n>: Address Comparator Access Type Register <n>

TRCACVR<n>: Address Comparator Value Register <n>

TRCAUTHSTATUS: Authentication Status Register

TRCAUXCTLR: Auxillary Control Register

TRCBBCCTRL: Branch Broadcast Control Register

TRCCCCTRLR: Cycle Count Control Register

TRCCIDCCTL0: Context Identifier Comparator Control Register 0

TRCCIDCCTL1: Context Identifier Comparator Control Register 1

TRCCIDCVRV<n>: Context Identifier Comparator Value Registers <n>

TRCCIDR0: Component Identification Register 0

TRCCIDR1: Component Identification Register 1

TRCCIDR2: Component Identification Register 2

TRCCIDR3: Component Identification Register 3

TRCCCLAIMCLR: Claim Tag Clear Register

TRCCCLAIMSET: Claim Tag Set Register

TRCCNCTRL<n>: Counter Control Register <n>

TRCCNTRLDRV<n>: Counter Reload Value Register <n>

TRCCNTVR<n>: Counter Value Register <n>

TRCCONFIGR: Trace Configuration Register

TRCDEVAFF: Device Affinity Register

TRCDEVARCH: Device Architecture Register

TRCDEVID: Device Configuration Register

TRCDEVID1: Device Configuration Register 1

TRCDEVID2: Device Configuration Register 2

TRCDEVTYPE: Device Type Register

TRCEVENTCTRL0: Event Control 0 Register

TRCEVENTCTRL1R: Event Control 1 Register
TRCEXITINSLR<n>: External Input Select Register <n>
TRCIDR0: ID Register 0
TRCIDR1: ID Register 1
TRCIDR10: ID Register 10
TRCIDR11: ID Register 11
TRCIDR12: ID Register 12
TRCIDR13: ID Register 13
TRCIDR2: ID Register 2
TRCIDR3: ID Register 3
TRCIDR4: ID Register 4
TRCIDR5: ID Register 5
TRCIDR6: ID Register 6
TRCIDR7: ID Register 7
TRCIDR8: ID Register 8
TRCIDR9: ID Register 9
TRCIMSPEC0: IMP DEF Register 0
TRCIMSPEC<n>: IMP DEF Register <n>
TRCITCTRL: Integration Mode Control Register
TRCLAR: Lock Access Register
TRCLSR: Lock Status Register
TRCOSLSR: Trace OS Lock Status Register
TRCPDCR: PowerDown Control Register
TRCPDSR: PowerDown Status Register
TRCPIDR0: Peripheral Identification Register 0
TRCPIDR1: Peripheral Identification Register 1
TRCPIDR2: Peripheral Identification Register 2
TRCPIDR3: Peripheral Identification Register 3
TRCPIDR4: Peripheral Identification Register 4
TRCPIDR5: Peripheral Identification Register 5
TRCPIDR6: Peripheral Identification Register 6
TRCPIDR7: Peripheral Identification Register 7
TRCPRGBCTL: Programming Control Register
TRCOCTRL: Q Element Control Register
TRCRSCTL<n>: Resource Selection Control Register <n>
TRCRSR: Resources Status Register
TRCSEQEV<n>: Sequencer State Transition Control Register <n>
**External System registers**

**TRCSEQSTEVTR**: Sequencer Reset Control Register

**TRCSEQOSTR**: Sequencer State Register

**TRCSSCCCR<n>**: Single-shot Comparator Control Register <n>

**TRCSSCSR<n>**: Single-shot Comparator Control Status Register <n>

**TRCSSPCICR<n>**: Single-shot Processing Element Comparator Input Control Register <n>

**TRCSTALLCTLR**: Stall Control Register

**TRCSATR**: Trace Status Register

**TRCSYNCR**: Synchronization Period Register

**TRCTRACEIDR**: Trace ID Register

**TRCTSCTRL**: Timestamp Control Register

**TRCVICTLR**: ViewInst Main Control Register

**TRCVIIECTLR**: ViewInst Include/Exclude Control Register

**TRCVIPCSSCTRL**: ViewInst Start/Stop PE Comparator Control Register

**TRCVISSCTRL**: ViewInst Start/Stop Control Register

**TRCVMIDCCTLR0**: Virtual Context Identifier Comparator Control Register 0

**TRCVMIDCCTLR1**: Virtual Context Identifier Comparator Control Register 1

**TRCVMIDCVR<n>**: Virtual Context Identifier Comparator Value Register <n>
Below are indexes for external registers in the following blocks:

- AMU
- CTI
- Debug
- ETF
- GIC CPU interface
- GIC Distributor
- GIC ITS control
- GIC ITS translation
- GIC Redistributor
- GIC Virtual CPU interface
- GIC Virtual interface control
- MPAM
- PMU
- RAS
- Timer

In the AMU block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000  + 8n</td>
<td>AMEVCNTR0&lt;n&gt;[31:0]</td>
<td>Activity Monitors Event Counter Registers 0</td>
</tr>
<tr>
<td>0x004  + 8n</td>
<td>AMEVCNTR0&lt;n&gt;[63:32]</td>
<td>Activity Monitors Event Counter Registers 0</td>
</tr>
<tr>
<td>0x100  + 8n</td>
<td>AMEVCNTR1&lt;n&gt;[31:0]</td>
<td>Activity Monitors Event Counter Registers 1</td>
</tr>
<tr>
<td>0x104  + 8n</td>
<td>AMEVCNTR1&lt;n&gt;[63:32]</td>
<td>Activity Monitors Event Counter Registers 1</td>
</tr>
<tr>
<td>0x400  + 4n</td>
<td>AMEVTYPER0&lt;n&gt;</td>
<td>Activity Monitors Event Type Registers 0</td>
</tr>
<tr>
<td>0x480  + 4n</td>
<td>AMEVTYPER1&lt;n&gt;</td>
<td>Activity Monitors Event Type Registers 1</td>
</tr>
<tr>
<td>0xC00</td>
<td>AMCNTENSET0</td>
<td>Activity Monitors Count Enable Set Register 0</td>
</tr>
<tr>
<td>0xC04</td>
<td>AMCNTENSET1</td>
<td>Activity Monitors Count Enable Set Register 1</td>
</tr>
<tr>
<td>0xC20</td>
<td>AMCNTENCLR0</td>
<td>Activity Monitors Count Enable Clear Register 0</td>
</tr>
<tr>
<td>0xC24</td>
<td>AMCNTENCLR1</td>
<td>Activity Monitors Count Enable Clear Register 1</td>
</tr>
<tr>
<td>0xCE0</td>
<td>AMCGCR</td>
<td>Activity Monitors Counter Group Configuration Register</td>
</tr>
<tr>
<td>0xE00</td>
<td>AMCFGR</td>
<td>Activity Monitors Configuration Register</td>
</tr>
<tr>
<td>0xE04</td>
<td>AMCR</td>
<td>Activity Monitors Control Register</td>
</tr>
<tr>
<td>0xE08</td>
<td>AMIIDR</td>
<td>Activity Monitors Implementation Identification Register</td>
</tr>
<tr>
<td>0xFA8</td>
<td>AMDEVAFF0</td>
<td>Activity Monitors Device Affinity Register 0</td>
</tr>
<tr>
<td>0xFAC</td>
<td>AMDEVAFF1</td>
<td>Activity Monitors Device Affinity Register 1</td>
</tr>
<tr>
<td>0xFB0</td>
<td>AMDEVARC</td>
<td>Activity Monitors Device Architecture Register</td>
</tr>
<tr>
<td>0xFC0</td>
<td>AMDEVTYPE</td>
<td>Activity Monitors Device Type Register</td>
</tr>
<tr>
<td>0xFD0</td>
<td>AMPIDR4</td>
<td>Activity Monitors Peripheral Identification Register 4</td>
</tr>
<tr>
<td>0xFE0</td>
<td>AMPIDR0</td>
<td>Activity Monitors Peripheral Identification Register 0</td>
</tr>
<tr>
<td>0xFE4</td>
<td>AMPIDR1</td>
<td>Activity Monitors Peripheral Identification Register 1</td>
</tr>
<tr>
<td>0xFE8</td>
<td>AMPIDR2</td>
<td>Activity Monitors Peripheral Identification Register 2</td>
</tr>
<tr>
<td>0xFEC</td>
<td>AMPIDR3</td>
<td>Activity Monitors Peripheral Identification Register 3</td>
</tr>
<tr>
<td>0xFF0</td>
<td>AMCIDR0</td>
<td>Activity Monitors Component Identification Register 0</td>
</tr>
<tr>
<td>0xFF4</td>
<td>AMCIDR1</td>
<td>Activity Monitors Component Identification Register 1</td>
</tr>
<tr>
<td>0xFF8</td>
<td>AMCIDR2</td>
<td>Activity Monitors Component Identification Register 2</td>
</tr>
<tr>
<td>0xFFC</td>
<td>AMCIDR3</td>
<td>Activity Monitors Component Identification Register 3</td>
</tr>
</tbody>
</table>
### In the CTI block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000</td>
<td>CTICONTROL</td>
<td>CTI Control register</td>
</tr>
<tr>
<td>0x010</td>
<td>CTIINTACK</td>
<td>CTI Output Trigger Acknowledge register</td>
</tr>
<tr>
<td>0x014</td>
<td>CTIAPPSET</td>
<td>CTI Application Trigger Set register</td>
</tr>
<tr>
<td>0x018</td>
<td>CTIAPPCLEAR</td>
<td>CTI Application Trigger Clear register</td>
</tr>
<tr>
<td>0x01C</td>
<td>CTIAPPPULSE</td>
<td>CTI Application Pulse register</td>
</tr>
<tr>
<td>0x020 + 4n</td>
<td>CIINEN&lt;n&gt;</td>
<td>CTI Input Trigger to Output Channel Enable registers</td>
</tr>
<tr>
<td>0x0A0 + 4n</td>
<td>CIOUTEN&lt;n&gt;</td>
<td>CTI Input Channel to Output Trigger Enable registers</td>
</tr>
<tr>
<td>0x130</td>
<td>CITRIGINSTATUS</td>
<td>CTI Trigger In Status register</td>
</tr>
<tr>
<td>0x134</td>
<td>CITRIGOUTSTATUS</td>
<td>CTI Trigger Out Status register</td>
</tr>
<tr>
<td>0x138</td>
<td>CTICHINSTATUS</td>
<td>CTI Channel In Status register</td>
</tr>
<tr>
<td>0x13C</td>
<td>CTICHOUTSTATUS</td>
<td>CTI Channel Out Status register</td>
</tr>
<tr>
<td>0x140</td>
<td>CTIGATE</td>
<td>CTI Channel Gate Enable register</td>
</tr>
<tr>
<td>0x144</td>
<td>ASICCTL</td>
<td>CTI External Multiplexer Control register</td>
</tr>
<tr>
<td>0x150</td>
<td>CTIDEVCTL</td>
<td>CTI Device Control register</td>
</tr>
<tr>
<td>0xF00</td>
<td>CTITCTRL</td>
<td>CTI Integration mode Control register</td>
</tr>
<tr>
<td>0xFA0</td>
<td>CTICLAIMSET</td>
<td>CTI CLAIM Tag Set register</td>
</tr>
<tr>
<td>0xFA4</td>
<td>CTICLAIMCLR</td>
<td>CTI CLAIM Tag Clear register</td>
</tr>
<tr>
<td>0xFA8</td>
<td>CTIDEVAFF0</td>
<td>CTI Device Affinity register 0</td>
</tr>
<tr>
<td>0xFAC</td>
<td>CTIDEVAFF1</td>
<td>CTI Device Affinity register 1</td>
</tr>
<tr>
<td>0xFB0</td>
<td>CTILAR</td>
<td>CTI Lock Access Register</td>
</tr>
<tr>
<td>0xFB4</td>
<td>CTILSR</td>
<td>CTI Lock Status Register</td>
</tr>
<tr>
<td>0xFB8</td>
<td>CTIAUTHSTATUS</td>
<td>CTI Authentication Status register</td>
</tr>
<tr>
<td>0xFC4</td>
<td>CTIDEVARCH</td>
<td>CTI Device Architecture register</td>
</tr>
<tr>
<td>0xFC0</td>
<td>CTIDEVID2</td>
<td>CTI Device ID register 2</td>
</tr>
<tr>
<td>0xFC4</td>
<td>CTIDEVID1</td>
<td>CTI Device ID register 1</td>
</tr>
<tr>
<td>0xFC8</td>
<td>CTIDEVID</td>
<td>CTI Device ID register 0</td>
</tr>
<tr>
<td>0xFF4</td>
<td>CTIDEVTYPE</td>
<td>CTI Device Type register</td>
</tr>
<tr>
<td>0xFE0</td>
<td>CTIPIDR4</td>
<td>CTI Peripheral Identification Register 4</td>
</tr>
<tr>
<td>0xFE0</td>
<td>CTIPIDR0</td>
<td>CTI Peripheral Identification Register 0</td>
</tr>
<tr>
<td>0xFE4</td>
<td>CTIPIDR1</td>
<td>CTI Peripheral Identification Register 1</td>
</tr>
<tr>
<td>0xFE8</td>
<td>CTIPIDR2</td>
<td>CTI Peripheral Identification Register 2</td>
</tr>
<tr>
<td>0xFE4</td>
<td>CTIPIDR3</td>
<td>CTI Peripheral Identification Register 3</td>
</tr>
<tr>
<td>0xFF0</td>
<td>CTICIDR0</td>
<td>CTI Component Identification Register 0</td>
</tr>
<tr>
<td>0xFF4</td>
<td>CTICIDR1</td>
<td>CTI Component Identification Register 1</td>
</tr>
<tr>
<td>0xFF8</td>
<td>CTICIDR2</td>
<td>CTI Component Identification Register 2</td>
</tr>
<tr>
<td>0xFFC</td>
<td>CTICIDR3</td>
<td>CTI Component Identification Register 3</td>
</tr>
</tbody>
</table>

### In the Debug block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x020</td>
<td>EDESR</td>
<td>External Debug Event Status Register</td>
</tr>
<tr>
<td>0x024</td>
<td>EDECR</td>
<td>External Debug Execution Control Register</td>
</tr>
<tr>
<td>0x030</td>
<td>EDWAR[31:0]</td>
<td>External Debug Watchpoint Address Register</td>
</tr>
<tr>
<td>0x034</td>
<td>EDWAR[63:32]</td>
<td>External Debug Watchpoint Address Register</td>
</tr>
<tr>
<td>0x080</td>
<td>DBGDTRRX_EL0</td>
<td>Debug Data Transfer Register, Receive</td>
</tr>
<tr>
<td>0x084</td>
<td>EDITR</td>
<td>External Debug Instruction Transfer Register</td>
</tr>
<tr>
<td>0x088</td>
<td>EDSR</td>
<td>External Debug Status and Control Register</td>
</tr>
<tr>
<td>Offset</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>0x08C</td>
<td>DBGDTKTX_EL0</td>
<td>Debug Data Transfer Register, Transmit</td>
</tr>
<tr>
<td>0x090</td>
<td>EDRCR</td>
<td>External Debug Reserve Control Register</td>
</tr>
<tr>
<td>0x094</td>
<td>EDACR</td>
<td>External Debug Auxiliary Control Register</td>
</tr>
<tr>
<td>0x098</td>
<td>EDECCR</td>
<td>External Debug Exception Catch Control Register</td>
</tr>
<tr>
<td>0x0A0</td>
<td>EDPCSR[31:0]</td>
<td>External Debug Program Counter Sample Register</td>
</tr>
<tr>
<td>0x0A4</td>
<td>EDCIDSR</td>
<td>External Debug Context ID Sample Register</td>
</tr>
<tr>
<td>0x0A8</td>
<td>EDVIDSR</td>
<td>External Debug Virtual Context Sample Register</td>
</tr>
<tr>
<td>0x0AC</td>
<td>EDPIDSR</td>
<td>External Debug Power/Reset Control Register</td>
</tr>
<tr>
<td>0x300</td>
<td>OSLAR_EL1</td>
<td>OS Lock Access Register</td>
</tr>
<tr>
<td>0x310</td>
<td>EDPRCR</td>
<td>External Debug Program Counter Sample Register</td>
</tr>
<tr>
<td>0x314</td>
<td>EDPRSR</td>
<td>External Debug Processor Status Register</td>
</tr>
<tr>
<td>0x400 + 16n</td>
<td>DBGVR&lt;n&gt;_EL1[63:0]</td>
<td>Debug Breakpoint Value Registers</td>
</tr>
<tr>
<td>0x408 + 16n</td>
<td>DBGBCR&lt;n&gt;_EL1</td>
<td>Debug Breakpoint Control Registers</td>
</tr>
<tr>
<td>0x800 + 16n</td>
<td>DBGWVR&lt;n&gt;_EL1[63:0]</td>
<td>Debug Watchpoint Value Registers</td>
</tr>
<tr>
<td>0x808 + 16n</td>
<td>DBGWCR&lt;n&gt;_EL1</td>
<td>Debug Watchpoint Control Registers</td>
</tr>
<tr>
<td>0xD00</td>
<td>MIDR_EL1</td>
<td>Main ID Register</td>
</tr>
<tr>
<td>0x20</td>
<td>EDPR[31:0]</td>
<td>External Debug Processor Feature Register</td>
</tr>
<tr>
<td>0x24</td>
<td>EDPFR[63:32]</td>
<td>External Debug Processor Feature Register</td>
</tr>
<tr>
<td>0x28</td>
<td>EDFR[31:0]</td>
<td>External Debug Feature Register</td>
</tr>
<tr>
<td>0x2C</td>
<td>EDFR[63:32]</td>
<td>External Debug Feature Register</td>
</tr>
<tr>
<td>0xD00</td>
<td>EDAA32PFR</td>
<td>External Debug AArch32 Processor Feature Register</td>
</tr>
<tr>
<td>0xF00</td>
<td>EDITCTRL</td>
<td>External Debug Integration mode Control register</td>
</tr>
<tr>
<td>0xFA0</td>
<td>DBGCLAIMSET_EL1</td>
<td>Debug CLAIM Tag Set register</td>
</tr>
<tr>
<td>0xFA4</td>
<td>DBGCLAIMCLR_EL1</td>
<td>Debug CLAIM Tag Clear register</td>
</tr>
<tr>
<td>0xFA8</td>
<td>EDDEVAFF0</td>
<td>External Debug Device Affinity register 0</td>
</tr>
<tr>
<td>0xFAC</td>
<td>EDDEVAFF1</td>
<td>External Debug Device Affinity register 1</td>
</tr>
<tr>
<td>0xFB0</td>
<td>EDLAR</td>
<td>External Debug Lock Access Register</td>
</tr>
<tr>
<td>0xFB4</td>
<td>EDLSR</td>
<td>External Debug Lock Status Register</td>
</tr>
<tr>
<td>0xFB8</td>
<td>DBGAUTHSTATUS_EL1</td>
<td>Debug Authentication Status register</td>
</tr>
<tr>
<td>0xFB0</td>
<td>EDDEVARCH</td>
<td>External Debug Device Architecture register</td>
</tr>
<tr>
<td>0xFC0</td>
<td>EDDEVID2</td>
<td>External Debug Device ID register 2</td>
</tr>
<tr>
<td>0xFC4</td>
<td>EDDEVID1</td>
<td>External Debug Device ID register 1</td>
</tr>
<tr>
<td>0xFC8</td>
<td>EDDEVID</td>
<td>External Debug Device ID register 0</td>
</tr>
<tr>
<td>0xFF0</td>
<td>EDDEVTYPE</td>
<td>External Debug Device Type register</td>
</tr>
<tr>
<td>0xFD0</td>
<td>EDPIDR4</td>
<td>External Debug Peripheral Identification Register 4</td>
</tr>
<tr>
<td>0xFE0</td>
<td>EDPIDR0</td>
<td>External Debug Peripheral Identification Register 0</td>
</tr>
<tr>
<td>0xFE4</td>
<td>EDPIDR1</td>
<td>External Debug Peripheral Identification Register 1</td>
</tr>
<tr>
<td>0xFE8</td>
<td>EDPIDR2</td>
<td>External Debug Peripheral Identification Register 2</td>
</tr>
<tr>
<td>0xFE4</td>
<td>EDPIDR3</td>
<td>External Debug Peripheral Identification Register 3</td>
</tr>
<tr>
<td>0xFF0</td>
<td>EDCIDR0</td>
<td>External Debug Component Identification Register 0</td>
</tr>
<tr>
<td>0xFF4</td>
<td>EDCIDR1</td>
<td>External Debug Component Identification Register 1</td>
</tr>
<tr>
<td>0xFF8</td>
<td>EDCIDR2</td>
<td>External Debug Component Identification Register 2</td>
</tr>
<tr>
<td>0xFFC</td>
<td>EDCIDR3</td>
<td>External Debug Component Identification Register 3</td>
</tr>
</tbody>
</table>

**In the ETE block:**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x004</td>
<td>TRCPRGCCTRL</td>
<td>Programming Control Register</td>
</tr>
<tr>
<td>0x00C</td>
<td>TRCSTATR</td>
<td>Trace Status Register</td>
</tr>
<tr>
<td>Offset</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>0x010</td>
<td>TRCCONFIGR</td>
<td>Trace Configuration Register</td>
</tr>
<tr>
<td>0x018</td>
<td>TRCAUXCTLR</td>
<td>Auxiliary Control Register</td>
</tr>
<tr>
<td>0x020</td>
<td>TRCEVENTCTL0R</td>
<td>Event Control 0 Register</td>
</tr>
<tr>
<td>0x024</td>
<td>TRCEVENTCTL1R</td>
<td>Event Control 1 Register</td>
</tr>
<tr>
<td>0x028</td>
<td>TRCRSR</td>
<td>Resources Status Register</td>
</tr>
<tr>
<td>0x02C</td>
<td>TRCSTALLCTLR</td>
<td>Stall Control Register</td>
</tr>
<tr>
<td>0x030</td>
<td>TRCTSCTR</td>
<td>Timestamp Control Register</td>
</tr>
<tr>
<td>0x034</td>
<td>TRCSYNCRPR</td>
<td>Synchronization Period Register</td>
</tr>
<tr>
<td>0x038</td>
<td>TRCCCTLR</td>
<td>Cycle Count Control Register</td>
</tr>
<tr>
<td>0x03C</td>
<td>TRCBBCTR</td>
<td>Branch Broadcast Control Register</td>
</tr>
<tr>
<td>0x040</td>
<td>TRCTRACEIDR</td>
<td>Trace ID Register</td>
</tr>
<tr>
<td>0x044</td>
<td>TRCOCTRL</td>
<td>Q Element Control Register</td>
</tr>
<tr>
<td>0x080</td>
<td>TRCVICTLR</td>
<td>ViewInst Main Control Register</td>
</tr>
<tr>
<td>0x084</td>
<td>TRCVIIECTLR</td>
<td>ViewInst Include/Exclude Control Register</td>
</tr>
<tr>
<td>0x088</td>
<td>TRCVISSCTLR</td>
<td>ViewInst Start/Stop Control Register</td>
</tr>
<tr>
<td>0x08C</td>
<td>TRCVIPSSCTLR</td>
<td>ViewInst Start/Stop PE Comparator Control Register</td>
</tr>
<tr>
<td>0x100 + 4n</td>
<td>TRCSEQEVR&lt;n&gt;</td>
<td>Sequencer State Transition Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x118</td>
<td>TRCSEQRSTEV R</td>
<td>Sequencer Reset Control Register</td>
</tr>
<tr>
<td>0x11C</td>
<td>TRCSEQSTR</td>
<td>Sequencer State Register</td>
</tr>
<tr>
<td>0x120 + 4n</td>
<td>TRCEXTINSEL R&lt;n&gt;</td>
<td>External Input Select Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x140 + 4n</td>
<td>TRCCNTBLDVR&lt;n&gt;</td>
<td>Counter Reload Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x150 + 4n</td>
<td>TRCCNTCTLR&lt;n&gt;</td>
<td>Counter Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x160 + 4n</td>
<td>TRCCNTVR&lt;n&gt;</td>
<td>Counter Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x180</td>
<td>TRCIDR8</td>
<td>ID Register 8</td>
</tr>
<tr>
<td>0x184</td>
<td>TRCIDR9</td>
<td>ID Register 9</td>
</tr>
<tr>
<td>0x188</td>
<td>TRCIDR10</td>
<td>ID Register 10</td>
</tr>
<tr>
<td>0x18C</td>
<td>TRCIDR11</td>
<td>ID Register 11</td>
</tr>
<tr>
<td>0x190</td>
<td>TRCIDR12</td>
<td>ID Register 12</td>
</tr>
<tr>
<td>0x194</td>
<td>TRCIDR13</td>
<td>ID Register 13</td>
</tr>
<tr>
<td>0x1C0</td>
<td>TRCIMSPEC0</td>
<td>IMP DEF Register 0</td>
</tr>
<tr>
<td>0x1C0 + 4n</td>
<td>TRCIMSPEC&lt;n&gt;</td>
<td>IMP DEF Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x1E0</td>
<td>TRCIDR0</td>
<td>ID Register 0</td>
</tr>
<tr>
<td>0x1E4</td>
<td>TRCIDR1</td>
<td>ID Register 1</td>
</tr>
<tr>
<td>0x1E8</td>
<td>TRCIDR2</td>
<td>ID Register 2</td>
</tr>
<tr>
<td>0x1EC</td>
<td>TRCIDR3</td>
<td>ID Register 3</td>
</tr>
<tr>
<td>0x1F0</td>
<td>TRCIDR4</td>
<td>ID Register 4</td>
</tr>
<tr>
<td>0x1F4</td>
<td>TRCIDR5</td>
<td>ID Register 5</td>
</tr>
<tr>
<td>0x1F8</td>
<td>TRCIDR6</td>
<td>ID Register 6</td>
</tr>
<tr>
<td>0x1FC</td>
<td>TRCIDR7</td>
<td>ID Register 7</td>
</tr>
<tr>
<td>0x200 + 4n</td>
<td>TRCRSCCTLR&lt;n&gt;</td>
<td>Resource Selection Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x280 + 4n</td>
<td>TRCSSSCCR&lt;n&gt;</td>
<td>Single-shot Comparator Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x2A0 + 4n</td>
<td>TRCSSCSR&lt;n&gt;</td>
<td>Single-shot Comparator Control Status Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x2C0 + 4n</td>
<td>TRCSSPCICR&lt;n&gt;</td>
<td>Single-shot Processing Element Comparator Input Control Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x304</td>
<td>TRCOSLSR</td>
<td>Trace OS Lock Status Register</td>
</tr>
<tr>
<td>0x310</td>
<td>TRCPDCR</td>
<td>PowerDown Control Register</td>
</tr>
<tr>
<td>0x314</td>
<td>TRCPDSR</td>
<td>PowerDown Status Register</td>
</tr>
<tr>
<td>0x400 + 8n</td>
<td>TRCACVR&lt;n&gt;</td>
<td>Address Comparator Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x480 + 8n</td>
<td>TRCACATR&lt;n&gt;</td>
<td>Address Comparator Access Type Register &lt;n&gt;</td>
</tr>
</tbody>
</table>
### Offset Table

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x600 + 8n</td>
<td>TRCCIDCVR&lt;n&gt;</td>
<td>Context Identifier Comparator Value Registers &lt;n&gt;</td>
</tr>
<tr>
<td>0x640 + 8n</td>
<td>TRCVMICVR&lt;n&gt;</td>
<td>Virtual Context Identifier Comparator Value Register &lt;n&gt;</td>
</tr>
<tr>
<td>0x680</td>
<td>TRCCIDCCTLR0</td>
<td>Context Identifier Comparator Control Register 0</td>
</tr>
<tr>
<td>0x684</td>
<td>TRCCIDCCTLR1</td>
<td>Context Identifier Comparator Control Register 1</td>
</tr>
<tr>
<td>0x688</td>
<td>TRCVMICCTRLR0</td>
<td>Virtual Context Identifier Comparator Control Register 0</td>
</tr>
<tr>
<td>0x68C</td>
<td>TRCVMICCTRLR1</td>
<td>Virtual Context Identifier Comparator Control Register 1</td>
</tr>
<tr>
<td>0xF00</td>
<td>TRCITCTRL</td>
<td>Integration Mode Control Register</td>
</tr>
<tr>
<td>0xFA0</td>
<td>TRCCLAIMSET</td>
<td>Claim Tag Set Register</td>
</tr>
<tr>
<td>0xFA4</td>
<td>TRCCLAIMCLR</td>
<td>Claim Tag Clear Register</td>
</tr>
<tr>
<td>0xFA8</td>
<td>TRCDEVAFF</td>
<td>Device Affinity Register</td>
</tr>
<tr>
<td>0xFB0</td>
<td>TRCLAR</td>
<td>Lock Access Register</td>
</tr>
<tr>
<td>0xFB4</td>
<td>TRCLSR</td>
<td>Lock Status Register</td>
</tr>
<tr>
<td>0xFB8</td>
<td>TRCAUTHSTATUS</td>
<td>Authentication Status Register</td>
</tr>
<tr>
<td>0xFC0</td>
<td>TRCDEVARCH</td>
<td>Device Architecture Register</td>
</tr>
<tr>
<td>0xFC4</td>
<td>TRCDEVID2</td>
<td>Device Configuration Register</td>
</tr>
<tr>
<td>0xFC8</td>
<td>TRCDEVID1</td>
<td>Device Configuration Register</td>
</tr>
<tr>
<td>0xFCC</td>
<td>TRCDEVTYPE</td>
<td>Device Type Register</td>
</tr>
<tr>
<td>0xFD0</td>
<td>TRCPIDR4</td>
<td>Peripheral Identification Register 4</td>
</tr>
<tr>
<td>0xFD4</td>
<td>TRCPIDR5</td>
<td>Peripheral Identification Register 5</td>
</tr>
<tr>
<td>0xFD8</td>
<td>TRCPIDR6</td>
<td>Peripheral Identification Register 6</td>
</tr>
<tr>
<td>0xFD8</td>
<td>TRCPIDR7</td>
<td>Peripheral Identification Register 7</td>
</tr>
<tr>
<td>0xFE0</td>
<td>TRCPIDR0</td>
<td>Peripheral Identification Register 0</td>
</tr>
<tr>
<td>0xFE4</td>
<td>TRCPIDR1</td>
<td>Peripheral Identification Register 1</td>
</tr>
<tr>
<td>0xFE8</td>
<td>TRCPIDR2</td>
<td>Peripheral Identification Register 2</td>
</tr>
<tr>
<td>0xFE8</td>
<td>TRCPIDR3</td>
<td>Peripheral Identification Register 3</td>
</tr>
<tr>
<td>0xFFF0</td>
<td>TRCCIDR0</td>
<td>Component Identification Register 0</td>
</tr>
<tr>
<td>0xFFF4</td>
<td>TRCCIDR1</td>
<td>Component Identification Register 1</td>
</tr>
<tr>
<td>0xFFF8</td>
<td>TRCCIDR2</td>
<td>Component Identification Register 2</td>
</tr>
<tr>
<td>0xFFF8</td>
<td>TRCCIDR3</td>
<td>Component Identification Register 3</td>
</tr>
</tbody>
</table>

### In the GIC CPU interface block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>GICC_CTLR</td>
<td>CPU Interface Control Register</td>
</tr>
<tr>
<td>0x0004</td>
<td>GICC_PMR</td>
<td>CPU Interface Priority Mask Register</td>
</tr>
<tr>
<td>0x0008</td>
<td>GICC_BPR</td>
<td>CPU Interface Binary Point Register</td>
</tr>
<tr>
<td>0x000C</td>
<td>GICC_IAR</td>
<td>CPU Interface Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>0x0010</td>
<td>GICC_EOIR</td>
<td>CPU Interface End Of Interrupt Register</td>
</tr>
<tr>
<td>0x0014</td>
<td>GICC_RPR</td>
<td>CPU Interface Running Priority Register</td>
</tr>
<tr>
<td>0x0018</td>
<td>GICC_HPPR</td>
<td>CPU Interface Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>0x001C</td>
<td>GICC_ABP</td>
<td>CPU Interface Aliased Binary Point Register</td>
</tr>
<tr>
<td>0x0020</td>
<td>GICC_AIAR</td>
<td>CPU Interface Aliased Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>0x0024</td>
<td>GICC_AEOIR</td>
<td>CPU Interface Aliased End Of Interrupt Register</td>
</tr>
<tr>
<td>0x0028</td>
<td>GICC_AHPPR</td>
<td>CPU Interface Aliased Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>0x002C</td>
<td>GICC_STATUSR</td>
<td>CPU Interface Status Register</td>
</tr>
<tr>
<td>0x002C</td>
<td>GICC_STATUSR</td>
<td>CPU Interface Status Register</td>
</tr>
<tr>
<td>0x00D0 + 4n</td>
<td>GICC_APR&lt;n&gt;</td>
<td>CPU Interface Active Priorities Registers</td>
</tr>
<tr>
<td>0x00E0 + 4n</td>
<td>GICC_NSAPR&lt;n&gt;</td>
<td>CPU Interface Non-secure Active Priorities Registers</td>
</tr>
<tr>
<td>Offset</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-----------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>0x00FC</td>
<td>GICC_IIDR</td>
<td>CPU Interface Identification Register</td>
</tr>
<tr>
<td>0x1000</td>
<td>GICC_DIR</td>
<td>CPU Interface Deactivate Interrupt Register</td>
</tr>
</tbody>
</table>

**In the GIC Distributor block:**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>GICD_CTLR</td>
<td>Distributor Control Register</td>
</tr>
<tr>
<td>0x0004</td>
<td>GICD_TYPER</td>
<td>Interrupt Controller Type Register</td>
</tr>
<tr>
<td>0x0008</td>
<td>GICD_IIDR</td>
<td>Distributor Implementer Identification Register</td>
</tr>
<tr>
<td>0x0010</td>
<td>GICD_STATUSR</td>
<td>Error Reporting Status Register</td>
</tr>
<tr>
<td>0x0010</td>
<td>GICD_STATUSR</td>
<td>Error Reporting Status Register</td>
</tr>
<tr>
<td>0x0040</td>
<td>GICD_SETSPI_NS</td>
<td>Set Non-secure SPI Pending Register</td>
</tr>
<tr>
<td>0x0048</td>
<td>GICD_CLRSPI_NS</td>
<td>Clear Non-secure SPI Pending Register</td>
</tr>
<tr>
<td>0x0050</td>
<td>GICD_SETSPI_SR</td>
<td>Set Secure SPI Pending Register</td>
</tr>
<tr>
<td>0x0058</td>
<td>GICD_CLRSPI_SR</td>
<td>Clear Secure SPI Pending Register</td>
</tr>
<tr>
<td>0x0080 + 4n</td>
<td>GICD_IGROUPR&lt;n&gt;</td>
<td>Interrupt Group Registers</td>
</tr>
<tr>
<td>0x0100 + 4n</td>
<td>GICD_ISENABLER&lt;n&gt;</td>
<td>Interrupt Set-Enable Registers</td>
</tr>
<tr>
<td>0x0180 + 4n</td>
<td>GICD_ICENABLER&lt;n&gt;</td>
<td>Interrupt Clear-Enable Registers</td>
</tr>
<tr>
<td>0x0200 + 4n</td>
<td>GICD_ISPENDR&lt;n&gt;</td>
<td>Interrupt Set-Pending Registers</td>
</tr>
<tr>
<td>0x0280 + 4n</td>
<td>GICD_ICPENDR&lt;n&gt;</td>
<td>Interrupt Clear-Pending Registers</td>
</tr>
<tr>
<td>0x0300 + 4n</td>
<td>GICD_ISACTIVER&lt;n&gt;</td>
<td>Interrupt Set-Active Registers</td>
</tr>
<tr>
<td>0x0380 + 4n</td>
<td>GICD_ICACTIVER&lt;n&gt;</td>
<td>Interrupt Clear-Active Registers</td>
</tr>
<tr>
<td>0x0400 + 4n</td>
<td>GICD_IPRIORITYR&lt;n&gt;</td>
<td>Interrupt Priority Registers</td>
</tr>
<tr>
<td>0x0800 + 4n</td>
<td>GICD_ITARGETSR&lt;n&gt;</td>
<td>Interrupt Processor Targets Registers</td>
</tr>
<tr>
<td>0x0C00 + 4n</td>
<td>GICD_ICFGR&lt;n&gt;</td>
<td>Interrupt Configuration Registers</td>
</tr>
<tr>
<td>0x0D00 + 4n</td>
<td>GICD_IGRPMODR&lt;n&gt;</td>
<td>Interrupt Group Modifier Registers</td>
</tr>
<tr>
<td>0x0E00 + 4n</td>
<td>GICD_NSACR&lt;n&gt;</td>
<td>Non-secure Access Control Registers</td>
</tr>
<tr>
<td>0x0F00</td>
<td>GICD_SGIR</td>
<td>Software Generated Interrupt Register</td>
</tr>
<tr>
<td>0x0F10 + 4n</td>
<td>GICD_SPENDSGIR&lt;n&gt;</td>
<td>SGI Clear-Pending Registers</td>
</tr>
<tr>
<td>0x0F20 + 4n</td>
<td>GICD_SPENDSGIR&lt;n&gt;</td>
<td>SGI Set-Pending Registers</td>
</tr>
<tr>
<td>0x1000 + 4n</td>
<td>GICD_IGROUPR&lt;n&gt;E</td>
<td>Interrupt Group Registers (extended SPI range)</td>
</tr>
<tr>
<td>0x1200 + 4n</td>
<td>GICD_ISENABLER&lt;n&gt;E</td>
<td>Interrupt Set-Enable Registers</td>
</tr>
<tr>
<td>0x1400 + 4n</td>
<td>GICD_ICENABLER&lt;n&gt;E</td>
<td>Interrupt Clear-Enable Registers</td>
</tr>
<tr>
<td>0x1600 + 4n</td>
<td>GICD_ISPENDR&lt;n&gt;E</td>
<td>Interrupt Set-Pending Registers (extended SPI range)</td>
</tr>
<tr>
<td>0x1800 + 4n</td>
<td>GICD_ICPENDR&lt;n&gt;E</td>
<td>Interrupt Clear-Pending Registers (extended SPI range)</td>
</tr>
<tr>
<td>0x1A00 + 4n</td>
<td>GICD_ISACTIVER&lt;n&gt;E</td>
<td>Interrupt Set-Active Registers (extended SPI range)</td>
</tr>
<tr>
<td>0x1C00 + 4n</td>
<td>GICD_ICACTIVER&lt;n&gt;E</td>
<td>Interrupt Clear-Active Registers (extended SPI range)</td>
</tr>
<tr>
<td>0x2000 + 4n</td>
<td>GICD_IPRIORITYR&lt;n&gt;E</td>
<td>Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.</td>
</tr>
<tr>
<td>0x3000 + 4n</td>
<td>GICD_ICFGR&lt;n&gt;E</td>
<td>Interrupt Configuration Registers (Extended SPI Range)</td>
</tr>
<tr>
<td>0x3400 + 4n</td>
<td>GICD_IGRPMODR&lt;n&gt;E</td>
<td>Interrupt Group Modifier Registers (extended SPI range)</td>
</tr>
<tr>
<td>0x3600 + 4n</td>
<td>GICD_NSACR&lt;n&gt;E</td>
<td>Non-secure Access Control Registers</td>
</tr>
<tr>
<td>0x6000 + 8n</td>
<td>GICD_IROUTER&lt;n&gt;</td>
<td>Interrupt Routing Registers</td>
</tr>
<tr>
<td>0x8000 + 8n</td>
<td>GICD_IROUTER&lt;n&gt;E</td>
<td>Interrupt Routing Registers (Extended SPI Range)</td>
</tr>
</tbody>
</table>

**In the GIC ITS control block:**

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>GITS_CTLR</td>
<td>ITS Control Register</td>
</tr>
<tr>
<td>0x0004</td>
<td>GITS_IIDR</td>
<td>ITS Identification Register</td>
</tr>
</tbody>
</table>
### In the GIC ITS translation block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0040</td>
<td>GITS_TRANSLATER</td>
<td>ITS Translation Register</td>
</tr>
</tbody>
</table>

### In the GIC Redistributor block:

<table>
<thead>
<tr>
<th>Frame</th>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RD_base</td>
<td>0x0000</td>
<td>GICR_CTLR</td>
<td>Redistributor Control Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0004</td>
<td>GICR_IIDR</td>
<td>Redistributor Implementer Identification Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0008</td>
<td>GICR_TYPER</td>
<td>Redistributor Type Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0010</td>
<td>GICR_STATUSR</td>
<td>Error Reporting Status Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0014</td>
<td>GICR_PARTIDR</td>
<td>Set PARTID and PMG Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0018</td>
<td>GICR_MPAMIDR</td>
<td>Report maximum PARTID and PMG Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x001C</td>
<td>GICR_PARTIDR</td>
<td>Set PARTID and PMG Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0040</td>
<td>GICR_SETLPIR</td>
<td>Set LPI Pending Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0048</td>
<td>GICR_CLRPLPIR</td>
<td>Clear LPI Pending Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0070</td>
<td>GICR_PROPBASER</td>
<td>Redistributor Properties Base Address Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x0078</td>
<td>GICR_PENDBASER</td>
<td>Redistributor LPI Pending Table Base Address Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x00A0</td>
<td>GICR_INVLPIR</td>
<td>Redistributor Invalidate LPI Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x00B0</td>
<td>GICR_INVALLR</td>
<td>Redistributor Invalidate All Register</td>
</tr>
<tr>
<td>RD_base</td>
<td>0x00C0</td>
<td>GICR_SYNC</td>
<td>Redistributor Synchronize Register</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0080</td>
<td>GICR_IGROUPR0</td>
<td>Interrupt Group Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0080 + 4n</td>
<td>GICR_IGROUPR&lt;n&gt;E</td>
<td>Interrupt Group Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0100</td>
<td>GICR_ISENABLER0</td>
<td>Interrupt Set-Enable Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0100 + 4n</td>
<td>GICR_ISENABLER&lt;n&gt;E</td>
<td>Interrupt Set-Enable Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0180</td>
<td>GICR_ICENABLER0</td>
<td>Interrupt Clear-Enable Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0180 + 4n</td>
<td>GICR_ICENABLER&lt;n&gt;E</td>
<td>Interrupt Clear-Enable Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0200</td>
<td>GICR_ISPENDR0</td>
<td>Interrupt Set-Pending Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0200 + 4n</td>
<td>GICR_ISPENDR&lt;n&gt;E</td>
<td>Interrupt Set-Pending Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0280</td>
<td>GICR_ICPENDR0</td>
<td>Interrupt Clear-Pending Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0280 + 4n</td>
<td>GICR_ICPENDR&lt;n&gt;E</td>
<td>Interrupt Clear-Pending Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0300</td>
<td>GICR_ISACTIVER0</td>
<td>Interrupt Set-Active Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0300 + 4n</td>
<td>GICR_ISACTIVER&lt;n&gt;E</td>
<td>Interrupt Set-Active Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0380</td>
<td>GICR_ICACTIVER0</td>
<td>Interrupt Clear-Active Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0380 + 4n</td>
<td>GICR_ICACTIVER&lt;n&gt;E</td>
<td>Interrupt Clear-Active Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0400</td>
<td>GICR_IPRIORITYR&lt;n&gt;</td>
<td>Interrupt Priority Registers</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0400 + 4n</td>
<td>GICR_IPRIORITYR&lt;n&gt;E</td>
<td>Interrupt Priority Registers (extended PPI range)</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0C00</td>
<td>GICR_ICFGR0</td>
<td>Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>SGI_base</td>
<td>0x0C00 + 4n</td>
<td>GICR_ICFGR&lt;n&gt;E</td>
<td>Interrupt configuration registers</td>
</tr>
</tbody>
</table>
### In the GIC Virtual CPU interface block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>GICV_CTLR</td>
<td>Virtual Machine Control Register</td>
</tr>
<tr>
<td>0x0004</td>
<td>GICV_PMR</td>
<td>Virtual Machine Priority Mask Register</td>
</tr>
<tr>
<td>0x0008</td>
<td>GICV_BPR</td>
<td>Virtual Machine Binary Point Register</td>
</tr>
<tr>
<td>0x000C</td>
<td>GICV_IAR</td>
<td>Virtual Machine Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>0x0010</td>
<td>GICV_EOIR</td>
<td>Virtual Machine End Of Interrupt Register</td>
</tr>
<tr>
<td>0x0014</td>
<td>GICV_RPR</td>
<td>Virtual Machine Running Priority Register</td>
</tr>
<tr>
<td>0x0018</td>
<td>GICV_HPPIR</td>
<td>Virtual Machine Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>0x001C</td>
<td>GICV_ABPR</td>
<td>Virtual Machine Aliased Binary Point Register</td>
</tr>
<tr>
<td>0x0020</td>
<td>GICV_AIAR</td>
<td>Virtual Machine Aliased Interrupt Acknowledge Register</td>
</tr>
<tr>
<td>0x0024</td>
<td>GICV_AEOIR</td>
<td>Virtual Machine Aliased End Of Interrupt Register</td>
</tr>
<tr>
<td>0x0028</td>
<td>GICV_AHPPIR</td>
<td>Virtual Machine Aliased Highest Priority Pending Interrupt Register</td>
</tr>
<tr>
<td>0x002C</td>
<td>GICV_STATUSR</td>
<td>Virtual Machine Error Reporting Status Register</td>
</tr>
<tr>
<td>0x00D0 + 4n</td>
<td>GICV_APR&lt;n&gt;</td>
<td>Virtual Machine Active Priorities Registers</td>
</tr>
<tr>
<td>0x00FC</td>
<td>GICV_IDDR</td>
<td>Virtual Machine CPU Interface Identification Register</td>
</tr>
<tr>
<td>0x1000</td>
<td>GICV_DIR</td>
<td>Virtual Machine Deactivate Interrupt Register</td>
</tr>
</tbody>
</table>

### In the GIC Virtual interface control block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>GICH_HCR</td>
<td>Hypervisor Control Register</td>
</tr>
<tr>
<td>0x0004</td>
<td>GICH_VTR</td>
<td>Virtual Type Register</td>
</tr>
<tr>
<td>0x0008</td>
<td>GICH_VMCR</td>
<td>Virtual Machine Control Register</td>
</tr>
<tr>
<td>0x0010</td>
<td>GICH_MISR</td>
<td>Maintenance Interrupt Status Register</td>
</tr>
<tr>
<td>0x0020</td>
<td>GICH_EISR</td>
<td>End Interrupt Status Register</td>
</tr>
<tr>
<td>0x0030</td>
<td>GICH_ELRSR</td>
<td>Empty List Register Status Register</td>
</tr>
<tr>
<td>0x00F0 + 4n</td>
<td>GICH_APR&lt;n&gt;</td>
<td>Active Priorities Registers</td>
</tr>
<tr>
<td>0x1000 + 4n</td>
<td>GICH_LR&lt;n&gt;</td>
<td>List Registers</td>
</tr>
</tbody>
</table>

### In the MPAM block:

<table>
<thead>
<tr>
<th>Frame</th>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0000</td>
<td>MPAMF_IDR</td>
<td>MPAM Features Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0018</td>
<td>MPAMF_IDDR</td>
<td>MPAM Implementation Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0020</td>
<td>MPAMF_AIDR</td>
<td>MPAM Architecture Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0028</td>
<td>MPAMF_IMPL_IDR</td>
<td>MPAM Implementation-Specific Partitioning Feature Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0030</td>
<td>MPAMF_CPOR_IDR</td>
<td>MPAM Features Cache Portion Partitioning ID register</td>
</tr>
<tr>
<td>Frame</td>
<td>Offset</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0038</td>
<td>MPAMF_CCAP_IDR</td>
<td>MPAM Features Cache Capacity Partitioning ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0040</td>
<td>MPAMF_MBW_IDR</td>
<td>MPAM Memory Bandwidth Partitioning Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0048</td>
<td>MPAMF_PRI_IDR</td>
<td>MPAM Priority Partitioning Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0050</td>
<td>MPAMF_PARTID_NRW_IDR</td>
<td>MPAM PARTID Narrowing ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0080</td>
<td>MPAMF_MSMON_IDR</td>
<td>MPAM Resource Monitoring Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0088</td>
<td>MPAMF_CSUMON_IDR</td>
<td>MPAM Features Cache Storage Usage Monitoring ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0090</td>
<td>MPAMF_MBWUMON_IDR</td>
<td>MPAM Features Memory Bandwidth Usage Monitoring ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x00F0</td>
<td>MPAMF_ECR</td>
<td>MPAM Error Control Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x00F8</td>
<td>MPAMF_ESR</td>
<td>MPAM Error Status Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0100</td>
<td>MPAMCFG_PART_SEL</td>
<td>MPAM Partition Configuration Selection Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0108</td>
<td>MPAMCFG_CMAX</td>
<td>MPAM Cache Maximum Capacity Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0200</td>
<td>MPAMCFG_MBW_MIN</td>
<td>MPAM Cache Maximum Capacity Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0208</td>
<td>MPAMCFG_MBW_MAX</td>
<td>MPAM Memory Bandwidth Maximum Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0220</td>
<td>MPAMCFG_MBW_WINWD</td>
<td>MPAM Memory Bandwidth Partitioning Window Width Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0400</td>
<td>MPAMCFG_PRI</td>
<td>MPAM Priority Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0500</td>
<td>MPAMCFG_MBW_PROP</td>
<td>MPAM Memory Bandwidth Proportional Stride Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0600</td>
<td>MPAMCFG_INTPARTID</td>
<td>MPAM Internal PARTID Narrowing Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0800</td>
<td>MSON_CFG_MON_SEL</td>
<td>MPAM Monitor Instance Selection Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0808</td>
<td>MSON_CPT_EVNT</td>
<td>MPAM Capture Event Generation Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0810</td>
<td>MSON_CFG_CSUFLT</td>
<td>MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0818</td>
<td>MSON_CFG_CSUCTL</td>
<td>MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0820</td>
<td>MSON_CFG_MBWUFLT</td>
<td>MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0828</td>
<td>MSON_CFG_MBWUCTL</td>
<td>MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0840</td>
<td>MSON_CSU</td>
<td>MPAM Cache Storage Usage Monitor Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0848</td>
<td>MSON_CSUCAPTURE</td>
<td>MPAM Cache Storage Usage Monitor Capture Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0860</td>
<td>MSON_MBWU</td>
<td>MPAM Memory Bandwidth Usage Monitor Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0868</td>
<td>MSON_MBWU_CAPTURE</td>
<td>MPAM Memory Bandwidth Usage Monitor Capture Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0880</td>
<td>MSON_MBWUL</td>
<td>MPAM Long Memory Bandwidth Usage Monitor Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x0890</td>
<td>MSON_MBWUL_CAPTURE</td>
<td>MPAM Long Memory Bandwidth Usage Monitor Capture Register</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x1000</td>
<td>MPAMCFG_CPBM</td>
<td>MPAM Cache Portion Bitmap Partition Configuration Register</td>
</tr>
<tr>
<td>Frame</td>
<td>Offset</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>-------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>MPAMF_BASE_ns</td>
<td>0x2000</td>
<td>MPAMCFG_MBW_PBM</td>
<td>MPAM Bandwidth Portion Bitmap Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0000</td>
<td>MPAMF_IDR</td>
<td>MPAM Features Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0008</td>
<td>MPAMF_SIDR</td>
<td>MPAM Features Secure Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0018</td>
<td>MPAMF_IIDR</td>
<td>MPAM Implementation Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0020</td>
<td>MPAMF_AIDR</td>
<td>MPAM Architecture Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0028</td>
<td>MPAMF_IMPL_IDR</td>
<td>MPAM Implementation-Specific Partitioning Feature Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0030</td>
<td>MPAMF_CPOR_IDR</td>
<td>MPAM Features Cache Portion Partitioning ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0038</td>
<td>MPAMF_CCAP_IDR</td>
<td>MPAM Features Cache Capacity Partitioning ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0040</td>
<td>MPAMF_MBW_IDR</td>
<td>MPAM Memory Bandwidth Partitioning Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0048</td>
<td>MPAMF_PRI_IDR</td>
<td>MPAM Priority Partitioning Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0050</td>
<td>MPAMF_PARTID_NRW_IDR</td>
<td>MPAM PARTID Narrowing ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0080</td>
<td>MPAMF_MSMON_IDR</td>
<td>MPAM Resource Monitoring Identification Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0088</td>
<td>MPAMF_CSUMON_IDR</td>
<td>MPAM Features Cache Storage Usage Monitoring ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0090</td>
<td>MPAMF_MBWUMON_IDR</td>
<td>MPAM Features Memory Usage Monitoring ID register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x00F0</td>
<td>MPAMF_ECR</td>
<td>MPAM Error Control Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x00F8</td>
<td>MPAMF_ESR</td>
<td>MPAM Error Status Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0100</td>
<td>MPAMCFG_PART_SEL</td>
<td>MPAM Partition Configuration Selection Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0108</td>
<td>MPAMCFG_CMAX</td>
<td>MPAM Cache Maximum Capacity Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0200</td>
<td>MPAMCFG_MBW_MIN</td>
<td>MPAM Cache Maximum Capacity Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0208</td>
<td>MPAMCFG_MBW_MAX</td>
<td>MPAM Memory Bandwidth Maximum Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0220</td>
<td>MPAMCFG_MBW_WINWD</td>
<td>MPAM Memory Bandwidth Partitioning Window Width Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0400</td>
<td>MPAMCFG_PRI</td>
<td>MPAM Priority Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0500</td>
<td>MPAMCFG_MBW_PROP</td>
<td>MPAM Memory Bandwidth Proportional Stride Partition Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0600</td>
<td>MPAMCFG_INTPARTID</td>
<td>MPAM Internal PARTID Narrowing Configuration Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0800</td>
<td>MSMON_CFG_MON_SEL</td>
<td>MPAM Monitor Instance Selection Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0808</td>
<td>MSMON_Capt_EVNT</td>
<td>MPAM Capture Event Generation Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0810</td>
<td>MSMON_CFG_CSU_FLT</td>
<td>MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0818</td>
<td>MSMON_CFG_CSU_CTL</td>
<td>MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0820</td>
<td>MSMON_CFG_MBWU_FLT</td>
<td>MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register</td>
</tr>
<tr>
<td>MPAMF_BASE_s</td>
<td>0x0828</td>
<td>MSMON_CFG_MBWU_CTL</td>
<td>MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register</td>
</tr>
</tbody>
</table>
### In the PMU block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000 + 8n</td>
<td>PMEVCNTR&lt;n&gt;_EL0</td>
<td>Performance Monitors Event Count Registers</td>
</tr>
<tr>
<td>0x0F8</td>
<td>PMCCNTR_EL0[31:0]</td>
<td>Performance Monitors Cycle Counter</td>
</tr>
<tr>
<td>0x0FC</td>
<td>PMCCNTR_EL0[63:32]</td>
<td>Performance Monitors Cycle Counter</td>
</tr>
<tr>
<td>0x200</td>
<td>PMPCSR[31:0]</td>
<td>Program Counter Sample Register</td>
</tr>
<tr>
<td>0x204</td>
<td>PMPCSR[63:32]</td>
<td>Program Counter Sample Register</td>
</tr>
<tr>
<td>0x208</td>
<td>PMCID1ISR</td>
<td>CONTEXTIDR_EL1 Sample Register</td>
</tr>
<tr>
<td>0x20C</td>
<td>PMVIDSR</td>
<td>VMID Sample Register</td>
</tr>
<tr>
<td>0x220</td>
<td>PMPCSR[31:0]</td>
<td>Program Counter Sample Register</td>
</tr>
<tr>
<td>0x224</td>
<td>PMPCSR[63:32]</td>
<td>Program Counter Sample Register</td>
</tr>
<tr>
<td>0x228</td>
<td>PMCID1SR</td>
<td>CONTEXTIDR_EL1 Sample Register</td>
</tr>
<tr>
<td>0x22C</td>
<td>PMCID2SR</td>
<td>CONTEXTIDR_EL2 Sample Register</td>
</tr>
<tr>
<td>0x400 + 4n</td>
<td>PMEVTYPE&lt;i&gt;_EL0</td>
<td>Performance Monitors Event Type Registers</td>
</tr>
<tr>
<td>0x47C</td>
<td>PMCCFILTR_EL0</td>
<td>Performance Monitors Cycle Counter Filter Register</td>
</tr>
<tr>
<td>0xC00</td>
<td>PMCNTENSET_EL0</td>
<td>Performance Monitors Count Enable Set register</td>
</tr>
<tr>
<td>0xC20</td>
<td>PMCNTENCLR_EL0</td>
<td>Performance Monitors Count Enable Clear register</td>
</tr>
<tr>
<td>0xC40</td>
<td>PMINTENSET_EL1</td>
<td>Performance Monitors Interrupt Enable Set register</td>
</tr>
<tr>
<td>0xC60</td>
<td>PMINTENCLR_EL1</td>
<td>Performance Monitors Interrupt Enable Clear register</td>
</tr>
<tr>
<td>0xC80</td>
<td>PMOVSCLR_EL0</td>
<td>Performance Monitors Overflow Flag Status Clear register</td>
</tr>
<tr>
<td>0xCA0</td>
<td>PMSWINC_EL0</td>
<td>Performance Monitors Software Increment register</td>
</tr>
<tr>
<td>0xCC0</td>
<td>PMOVSSSET_EL0</td>
<td>Performance Monitors Overflow Flag Status Set register</td>
</tr>
<tr>
<td>0xE00</td>
<td>PMCFGFR</td>
<td>Performance Monitors Configuration Register</td>
</tr>
<tr>
<td>0xE04</td>
<td>PMCR_EL0</td>
<td>Performance Monitors Control Register</td>
</tr>
<tr>
<td>0xE20</td>
<td>PMCEID0</td>
<td>Performance Monitors Common Event Identification register 0</td>
</tr>
<tr>
<td>0xE24</td>
<td>PMCEID1</td>
<td>Performance Monitors Common Event Identification register 1</td>
</tr>
<tr>
<td>0xE28</td>
<td>PMCEID2</td>
<td>Performance Monitors Common Event Identification register 2</td>
</tr>
<tr>
<td>0xE2C</td>
<td>PMCEID3</td>
<td>Performance Monitors Common Event Identification register 3</td>
</tr>
<tr>
<td>0xE40</td>
<td>PMMIR</td>
<td>Performance Monitors Machine Identification Register</td>
</tr>
<tr>
<td>0xF00</td>
<td>PMITCTRL</td>
<td>Performance Monitors Integration mode Control register</td>
</tr>
<tr>
<td>0xFA8</td>
<td>PMDEVAFF0</td>
<td>Performance Monitors Device Affinity register 0</td>
</tr>
<tr>
<td>0xFAC</td>
<td>PMDEVAFF1</td>
<td>Performance Monitors Device Affinity register 1</td>
</tr>
<tr>
<td>0xFB0</td>
<td>PMLAR</td>
<td>Performance Monitors Lock Access Register</td>
</tr>
<tr>
<td>0xFB4</td>
<td>PMLSR</td>
<td>Performance Monitors Lock Status Register</td>
</tr>
</tbody>
</table>
### Offset | Name | Description
--- | --- | ---
0xFB8 | PMAUTHSTATUS | Performance Monitors Authentication Status register
0xFBC | PMDEVARCH | Performance Monitors Device Architecture register
0xFC8 | PMDEVID | Performance Monitors Device ID register
0xFCC | PMDEVTYPE | Performance Monitors Device Type register
0xFD0 | PMPIDR4 | Performance Monitors Peripheral Identification Register 4
0xFE0 | PMPIDR0 | Performance Monitors Peripheral Identification Register 0
0xFE4 | PMPIDR1 | Performance Monitors Peripheral Identification Register 1
0xFE8 | PMPIDR2 | Performance Monitors Peripheral Identification Register 2
0xFF0 | PMPIDR3 | Performance Monitors Peripheral Identification Register 3
0xFF4 | PMCIDR0 | Performance Monitors Component Identification Register 0
0xFF8 | PMCIDR1 | Performance Monitors Component Identification Register 1
0xFFC | PMCIDR2 | Performance Monitors Component Identification Register 2
0xFFFF | PMCIDR3 | Performance Monitors Component Identification Register 3

### In the RAS block:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x000 + 64n</td>
<td>ERR&lt;n&gt;FR</td>
<td>Error Record Feature Register</td>
</tr>
<tr>
<td>0x008 + 64n</td>
<td>ERR&lt;n&gt;CTLR</td>
<td>Error Record Control Register</td>
</tr>
<tr>
<td>0x010 + 64n</td>
<td>ERR&lt;n&gt;STATUS</td>
<td>Error Record Primary Status Register</td>
</tr>
<tr>
<td>0x018 + 64n</td>
<td>ERR&lt;n&gt;ADDR</td>
<td>Error Record Address Register</td>
</tr>
<tr>
<td>0x020 + 64n</td>
<td>ERR&lt;n&gt;MISC0</td>
<td>Error Record Miscellaneous Register 0</td>
</tr>
<tr>
<td>0x028 + 64n</td>
<td>ERR&lt;n&gt;MISC1</td>
<td>Error Record Miscellaneous Register 1</td>
</tr>
<tr>
<td>0x030 + 64n</td>
<td>ERR&lt;n&gt;MISC2</td>
<td>Error Record Miscellaneous Register 2</td>
</tr>
<tr>
<td>0x038 + 64n</td>
<td>ERR&lt;n&gt;MISC3</td>
<td>Error Record Miscellaneous Register 3</td>
</tr>
<tr>
<td>0x800 + 64n</td>
<td>ERR&lt;n&gt;PFGF</td>
<td>Pseudo-fault Generation Feature Register</td>
</tr>
<tr>
<td>0x808 + 64n</td>
<td>ERR&lt;n&gt;PFGCTL</td>
<td>Pseudo-fault Generation Control Register</td>
</tr>
<tr>
<td>0x810 + 64n</td>
<td>ERR&lt;n&gt;PFGCDN</td>
<td>Pseudo-fault Generation Countdown Register</td>
</tr>
<tr>
<td>0xE00</td>
<td>ERRGSR</td>
<td>Error Group Status Register</td>
</tr>
<tr>
<td>0xE10</td>
<td>ERRIIDR</td>
<td>Implementation Identification Register</td>
</tr>
<tr>
<td>0xE80</td>
<td>ERRFHICR0</td>
<td>Fault-Handling Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>0xE80 + 8n</td>
<td>ERRRIQCR&lt;n&gt;</td>
<td>Generic Error Interrupt Configuration Register</td>
</tr>
<tr>
<td>0xE88</td>
<td>ERRFHICR1</td>
<td>Fault-Handling Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>0xE8C</td>
<td>ERRFHICR2</td>
<td>Fault-Handling Interrupt Configuration Register 2</td>
</tr>
<tr>
<td>0xE90</td>
<td>ERRERICR0</td>
<td>Error Recovery Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>0xE98</td>
<td>ERRERICR1</td>
<td>Error Recovery Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>0xE9C</td>
<td>ERRERICR2</td>
<td>Error Recovery Interrupt Configuration Register 2</td>
</tr>
<tr>
<td>0xEA0</td>
<td>ERRRCR0</td>
<td>Critical Error Interrupt Configuration Register 0</td>
</tr>
<tr>
<td>0xEA8</td>
<td>ERRRCR1</td>
<td>Critical Error Interrupt Configuration Register 1</td>
</tr>
<tr>
<td>0xEAC</td>
<td>ERRRCR2</td>
<td>Critical Error Interrupt Configuration Register 2</td>
</tr>
<tr>
<td>0xEF8</td>
<td>ERRRIROS</td>
<td>Error Interrupt Status Register</td>
</tr>
<tr>
<td>0xFA8</td>
<td>ERRDEVAFF</td>
<td>Device Affinity Register</td>
</tr>
<tr>
<td>0xFB0</td>
<td>ERRDEVAR</td>
<td>Device Architecture Register</td>
</tr>
<tr>
<td>0xFC8</td>
<td>ERRDEVID</td>
<td>Device Configuration Register</td>
</tr>
<tr>
<td>0xFD0</td>
<td>ERRPIDR4</td>
<td>Peripheral Identification Register 4</td>
</tr>
<tr>
<td>0xFE0</td>
<td>ERRPIDR0</td>
<td>Peripheral Identification Register 0</td>
</tr>
<tr>
<td>0xFE4</td>
<td>ERRPIDR1</td>
<td>Peripheral Identification Register 1</td>
</tr>
<tr>
<td>0xFE8</td>
<td>ERRPIDR2</td>
<td>Peripheral Identification Register 2</td>
</tr>
<tr>
<td>0xFE0</td>
<td>ERRPIDR3</td>
<td>Peripheral Identification Register 3</td>
</tr>
</tbody>
</table>
### Offset | Name | Description
--- | --- | ---
0xFF0 | ERRCIDR0 | Component Identification Register 0
0xFF4 | ERRCIDR1 | Component Identification Register 1
0xFF8 | ERRCIDR2 | Component Identification Register 2
0xFFC | ERRCIDR3 | Component Identification Register 3

### In the Timer block:

<table>
<thead>
<tr>
<th>Frame</th>
<th>Offset</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTBaseN</td>
<td>0x000</td>
<td>CNTPCT[31:0]</td>
<td>Counter-timer Physical Count</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x004</td>
<td>CNTPCT[63:32]</td>
<td>Counter-timer Physical Count</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x008</td>
<td>CNTVCT[31:0]</td>
<td>Counter-timer Virtual Count</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x00C</td>
<td>CNTVCT[63:32]</td>
<td>Counter-timer Virtual Count</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x010</td>
<td>CNTFRO</td>
<td>Counter-timer Frequency</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x014</td>
<td>CNT0ELACR</td>
<td>Counter-timer ELO Access Control Register</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x018</td>
<td>CNTVOFF[31:0]</td>
<td>Counter-timer Virtual Offset</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x01C</td>
<td>CNTVOFF[63:32]</td>
<td>Counter-timer Virtual Offset</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x020</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x024</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x028</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x02C</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x030</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x034</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x038</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0x03C</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>CNTBaseN</td>
<td>0xFD0 + 4n</td>
<td>CounterID&lt;n&gt;</td>
<td>Counter ID registers</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0x000</td>
<td>CNTFRO</td>
<td>Counter-timer Frequency</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0x004</td>
<td>CNTNSAR</td>
<td>Counter-timer Non-secure Access Register</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0x008</td>
<td>CNTIDS</td>
<td>Counter-timer Timer ID Register</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0x040 + 4n</td>
<td>CNTACR&lt;n&gt;</td>
<td>Counter-timer Access Control Registers</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0x080 + 8n</td>
<td>CNTVOFF&lt;n&gt;[31:0]</td>
<td>Counter-timer Virtual Offsets</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0x084 + 8n</td>
<td>CNTVOFF&lt;n&gt;[63:32]</td>
<td>Counter-timer Virtual Offsets</td>
</tr>
<tr>
<td>CNTCTLBase</td>
<td>0xFD0 + 4n</td>
<td>CounterID&lt;n&gt;</td>
<td>Counter ID registers</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x000</td>
<td>CNTCR</td>
<td>Counter Control Register</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x004</td>
<td>CNTSR</td>
<td>Counter Status Register</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x008</td>
<td>CNTVC[63:0]</td>
<td>Counter Count Value register</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x020</td>
<td>CNTFID0</td>
<td>Counter Frequency ID</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x020 + 4n</td>
<td>CNTFID&lt;n&gt;</td>
<td>Counter Frequency IDs, n &gt; 0</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x10</td>
<td>CNTSCR</td>
<td>Counter Scale Register</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0x1C</td>
<td>CNTID</td>
<td>Counter Identification Register</td>
</tr>
<tr>
<td>CNTControlBase</td>
<td>0xFD0 + 4n</td>
<td>CounterID&lt;n&gt;</td>
<td>Counter ID registers</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x000</td>
<td>CNTPCT[31:0]</td>
<td>Counter-timer Physical Count</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x004</td>
<td>CNTPCT[63:32]</td>
<td>Counter-timer Physical Count</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x008</td>
<td>CNTVCT[31:0]</td>
<td>Counter-timer Virtual Count</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x00C</td>
<td>CNTVCT[63:32]</td>
<td>Counter-timer Virtual Count</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x010</td>
<td>CNTFRO</td>
<td>Counter-timer Frequency</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x020</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x024</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x028</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>CNTTEL0BaseN</td>
<td>0x02C</td>
<td>CNTPV[63:32]</td>
<td>Counter-timer Physical Timer CompareValue</td>
</tr>
<tr>
<td>Frame</td>
<td>Offset</td>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------</td>
<td>-----------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>CNTEL0BaseN</td>
<td>0x030</td>
<td>CNTV_CVAL[31:0]</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>CNTEL0BaseN</td>
<td>0x034</td>
<td>CNTV_CVAL[63:32]</td>
<td>Counter-timer Virtual Timer CompareValue</td>
</tr>
<tr>
<td>CNTEL0BaseN</td>
<td>0x038</td>
<td>CNTV_TVAL</td>
<td>Counter-timer Virtual Timer TimerValue</td>
</tr>
<tr>
<td>CNTEL0BaseN</td>
<td>0x03C</td>
<td>CNTV_CTL</td>
<td>Counter-timer Virtual Timer Control</td>
</tr>
<tr>
<td>CNTEL0BaseN</td>
<td>0xFD0 + 4n</td>
<td>CounterID&lt;n&gt;</td>
<td>Counter ID registers</td>
</tr>
<tr>
<td>CNTReadBase</td>
<td>0x000</td>
<td>CNTCV[63:0]</td>
<td>Counter Count Value register</td>
</tr>
<tr>
<td>CNTReadBase</td>
<td>0xFD0 + 4n</td>
<td>CounterID&lt;n&gt;</td>
<td>Counter ID registers</td>
</tr>
</tbody>
</table>
AMCFGR, Activity Monitors Configuration Register

The AMCFGR characteristics are:

**Purpose**

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR is applicable to both the architected and the auxiliary counter groups.

**Configuration**

External register AMCFGR bits [31:0] are architecturally mapped to AArch64 System register AMCFGR_EL0[31:0].

External register AMCFGR bits [31:0] are architecturally mapped to AArch32 System register AMCFGR[31:0].

The power domain of AMCFGR is IMPLEMENTATION DEFINED.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCFGR are RES0.

**Attributes**

AMCFGR is a 32-bit register.

**Field descriptions**

The AMCFGR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NCG | RES0 | HDBG | RAZ | SIZE | N |

**NCG, bits [31:28]**

Defines the number of counter groups.

The number of implemented counter groups is defined as [AMCFGR.NCG + 1].

If the number of implemented auxiliary activity monitor event counters is zero, this field has a value of 0b0000. Otherwise, this field has a value of 0b0001.

**Bits [27:25]**

Reserved, RES0.

**HDBG, bit [24]**

Halt-on-debug supported.

From Armv8, this feature must be supported, and so this bit is 0b1.

<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>AMCR.HDBG is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>AMCR.HDBG is read/write.</td>
</tr>
</tbody>
</table>
Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of activity monitor event counters.

The size of the activity monitor event counters implemented by the Activity Monitors Extension is defined as [AMCFGR.SIZE + 1].

From Armv8, the counters are 64-bit, and so this field is 0b111111.

Note

Software also uses this field to determine the spacing of counters in the memory-map. From Armv8, the counters are at doubleword-aligned addresses.

N, bits [7:0]

Defines the number of activity monitor event counters.

The total number of counters implemented in all groups by the Activity Monitors Extension is defined as [AMCFGR.N + 1].

Accessing the AMCFGR

AMCFGR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xE00</td>
<td>AMCFGR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The AMCGCR characteristics are:

**Purpose**

Provides information on the number of activity monitor event counters implemented within each counter group.

**Configuration**

External register AMCGCR bits [31:0] are architecturally mapped to AArch64 System register AMCGCR_EL0[31:0].

External register AMCGCR bits [31:0] are architecturally mapped to AArch32 System register AMCGCR[31:0].

The power domain of AMCGCR is IMPLEMENTATION DEFINED.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCGCR are RES0.

**Attributes**

AMCGCR is a 32-bit register.

**Field descriptions**

The AMCGCR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>CG1NC</td>
<td>CG0NC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**CG1NC, bits [15:8]**

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In AMUv1, the permitted range of values is 0 to 16.

**CG0NC, bits [7:0]**

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

In AMUv1, the value of this field is 4.

**Accessing the AMCGCR**

AMCGCR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xCE0</td>
<td>AMCGCR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMCIDR0, Activity Monitors Component Identification Register 0

The AMCIDR0 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Component identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMCIDR0 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCIDR0 are RES0.

**Attributes**

AMCIDR0 is a 32-bit register.

**Field descriptions**

The AMCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>27-8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7-0</td>
<td>PRMBL_0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_0, bits [7:0]**

Preamble. Must read as 0x0D.

**Accessing the AMCIDR0**

**AMCIDR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFF0</td>
<td>AMCIDR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMCIDR1, Activity Monitors Component Identification Register 1

The AMCIDR1 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Component identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMCIDR1 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCIDR1 are RES0.

**Attributes**

AMCIDR1 is a 32-bit register.

**Field descriptions**

The AMCIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**CLASS, bits [7:4]**

Component class. Reads as 0x9, CoreSight component.

**PRMBL_1, bits [3:0]**

Preamble. Reads as 0x0.

**Accessing the AMCIDR1**

AMCIDR1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFF4</td>
<td>AMCIDR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMCIDR2, Activity Monitors Component Identification Register 2

The AMCIDR2 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Component identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMCIDR2 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCIDR2 are RES0.

**Attributes**

AMCIDR2 is a 32-bit register.

**Field descriptions**

The AMCIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>28-10</td>
<td>PRMBL_2, bits [7:0]</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_2, bits [7:0]**

Preamble. Reads as 0x05.

**Accessing the AMCIDR2**

AMCIDR2 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFF8</td>
<td>AMCIDR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMCIDR3, Activity Monitors Component Identification Register 3

The AMCIDR3 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Component identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMCIDR3 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCIDR3 are RES0.

**Attributes**

AMCIDR3 is a 32-bit register.

**Field descriptions**

The AMCIDR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 | PRMBL_3 |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble. Reads as 0xB1.

**Accessing the AMCIDR3**

**AMCIDR3 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFFC</td>
<td>AMCIDR3</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0 characteristics are:

**Purpose**

Disable control bits for the architected event counters, AMEVCTR0<n>.

**Configuration**

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR0_EL0[31:0].

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR0[31:0].

The power domain of AMCNTENCLR0 is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0 are RES0.

**Attributes**

AMCNTENCLR0 is a 32-bit register.

**Field descriptions**

The AMCNTENCLR0 bit assignments are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;, bit [n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
</tr>
<tr>
<td>---------------</td>
</tr>
</tbody>
</table>

**P<n>, bit [n], for n = 0 to 31**

Activity monitor event counter disable bit for AMEVCTR0<n>.

Bits [31:N] are RAZ/WI. N is the value in AMCGCR.CG0NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCTR0&lt;n&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCTR0&lt;n&gt; is enabled. When written, disables AMEVCTR0&lt;n&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Accessing the AMCNTENCLR0**

AMCNTENCLR0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xC20</td>
<td>AMCNTENSET0</td>
</tr>
</tbody>
</table>
Accesses on this interface are RO.
AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1 characteristics are:

**Purpose**

Disable control bits for the auxiliary activity monitors event counters, AMEVCNTR1<n>.

**Configuration**

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENCLR1_EL0[31:0].

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENCLR1[31:0].

The power domain of AMCNTENCLR1 is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1 are RES0.

**Attributes**

AMCNTENCLR1 is a 32-bit register.

**Field descriptions**

The AMCNTENCLR1 bit assignments are:

```
  31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
         P<n>, bit [n]
```

**P<n>, bit [n], for n = 0 to 31**

Activity monitor event counter disable bit for AMEVCNTR1<n>.

Bits [31:N] are RAZ/WI. N is the value in AMCGCR.CG1NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR1&lt;n&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR1&lt;n&gt; is enabled. When written, disable AMEVCNTR1&lt;n&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Accessing the AMCNTENCLR1**

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENCLR1 are CONSTRAINED UNPREDICTABLE, and accesses to the register behave as RAZ/WI.

**Note**


The number of auxiliary activity monitor event counters implemented is zero exactly when AMCFGR.NCG == 0b0000.

**AMCNTENCLR1 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xC24</td>
<td>AMCNTENCLR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
AMCNTENSET0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0 characteristics are:

**Purpose**

Enable control bits for the architected activity monitors event counters, AMEVCNTR0<\text{n}>

**Configuration**

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET0_EL0[31:0].

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET0[31:0].

The power domain of AMCNTENSET0 is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0 are RES0.

**Attributes**

AMCNTENSET0 is a 32-bit register.

**Field descriptions**

The AMCNTENSET0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>P&lt;n&gt; bit [n]</td>
<td></td>
</tr>
</tbody>
</table>

P<n>, bit [n], for n = 0 to 31

Activity monitor event counter enable bit for AMEVCNTR0<\text{n}>

Bits [31:N] are RAZ/WI. N is the value in AMCGCR.CG0NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCNTR0&lt;\text{n}&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCNTR0&lt;\text{n}&gt; is enabled. When written, enables AMEVCNTR0&lt;\text{n}&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Accessing the AMCNTENSET0**

AMCNTENSET0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xC00</td>
<td>AMCNTENSET0</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RO**.
AMCNTENSET1, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1 characteristics are:

**Purpose**

Enable control bits for the auxiliary activity monitors event counters, AMEVCTR1<n>.

**Configuration**

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register AMCNTENSET1_EL0[31:0].

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch32 System register AMCNTENSET1[31:0].

The power domain of AMCNTENSET1 is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1 are RES0.

**Attributes**

AMCNTENSET1 is a 32-bit register.

**Field descriptions**

The AMCNTENSET1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| P<n>, bit [n] |

**P<n>, bit [n], for n = 0 to 31**

Activity monitor event counter enable bit for AMEVCTR1<n>.

Bits [31:N] are RAZ/WI. N is the value in AMCGCR.CG1NC.

Possible values of each bit are:

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that AMEVCTR1&lt;n&gt; is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that AMEVCTR1&lt;n&gt; is enabled. When written, enables AMEVCTR1&lt;n&gt;.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Accessing the AMCNTENSET1**

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENSET1 are CONSTRAINED UNPREDICTABLE, and accesses to the register behave as RAZ/WI.

**Note**
The number of auxiliary activity monitor counters implemented is zero exactly when `AMCFG.NCG == 0b0000`.

**AMCNTENSET1 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xC04</td>
<td>AMCNTENSET1</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
**AMCR, Activity Monitors Control Register**

The AMCR characteristics are:

### Purpose

Global control register for the activity monitors implementation. AMCR is applicable to both the architected and the auxiliary counter groups.

### Configuration

External register AMCR bits [31:0] are architecturally mapped to AArch64 System register AMCR_EL0[31:0].

External register AMCR bits [31:0] are architecturally mapped to AArch32 System register AMCR[31:0].

The power domain of AMCR is IMPLEMENTATION DEFINED.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMCR are RES0.

### Attributes

AMCR is a 32-bit register.

### Field descriptions

The AMCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>HDBG</td>
<td>This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:11]**

Reserved, RES0.

**HDBG, bit [10]**

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Activity monitors do not halt counting when the PE is halted in Debug state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Activity monitors halt counting when the PE is halted in Debug state.</td>
</tr>
</tbody>
</table>

**Bits [9:0]**

Reserved, RAZ/WI.

### Accessing the AMCR

AMCR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xE04</td>
<td>AMCR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMDEVAFF0, Activity Monitors Device Affinity Register 0

The AMDEVAFF0 characteristics are:

**Purpose**

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

**Configuration**

The power domain of AMDEVAFF0 is **IMPLEMENTATION DEFINED**.

Implementation of this register is **OPTIONAL**.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMDEVAFF0 are **RES0**.

**Attributes**

AMDEVAFF0 is a 32-bit register.

**Field descriptions**

The AMDEVAFF0 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>MPIDR_EL1 low half</td>
</tr>
</tbody>
</table>

**MPIDR_EL1 low half, bits [31:0]**

MPIDR_EL1 low half. Read-only copy of the low half of MPIDR_EL1, as seen from the highest implemented Exception level.

**Accessing the AMDEVAFF0**

AMDEVAFF0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFA8</td>
<td>AMDEVAFF0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The AMDEVAFF1 characteristics are:

**Purpose**

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

**Configuration**

The power domain of AMDEVAFF1 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMDEVAFF1 are RES0.

**Attributes**

AMDEVAFF1 is a 32-bit register.

**Field descriptions**

The AMDEVAFF1 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>MPIDR_EL1 high half, bits [31:0]</td>
</tr>
</tbody>
</table>

MPIDR_EL1 high half. Read-only copy of the high half of MPIDR_EL1, as seen from the highest implemented Exception level.

**Accessing the AMDEVAFF1**

AMDEVAFF1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFAC</td>
<td>AMDEVAFF1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The AMDEVARCH characteristics are:

**Purpose**

Identifies the programmers' model architecture of the AMU component.

**Configuration**

The power domain of AMDEVARCH is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMDEVARCH are RES0.

**Attributes**

AMDEVARCH is a 32-bit register.

**Field descriptions**

The AMDEVARCH bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>ARCHITECT</th>
<th>PRESENT</th>
<th>REVISION</th>
<th>ARCHID</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
</tr>
</tbody>
</table>

**ARCHITECT, bits [31:21]**

Defines the architecture of the component. For AMU, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

**PRESENT, bit [20]**

When set to 1, indicates that the DEVARCH is present.

This field is 1 in Armv8.

**REVISION, bits [19:16]**

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

<table>
<thead>
<tr>
<th>REVISION</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Architecture revision is AMUv1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**ARCHID, bits [15:0]**

Defines this part to be an AMU component. For architectures defined by Arm this is further subdivided.

For AMU:
• Bits [15:12] are the architecture version, 0x0.
• Bits [11:0] are the architecture part number, 0xA66.

This corresponds to AMU architecture version AMUv1.

**Accessing the AMDEVARCH**

AMDEVARCH can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFBC</td>
<td>AMDEVARCH</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
AMDEVTYPE, Activity Monitors Device Type Register

The AMDEVTYPE characteristics are:

**Purpose**

Indicates to a debugger that this component is part of a PE's performance monitor interface.

**Configuration**

The power domain of AMDEVTYPE is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMDEVTYPE are RES0.

**Attributes**

AMDEVTYPE is a 32-bit register.

**Field descriptions**

The AMDEVTYPE bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>27-8</td>
<td>SUB</td>
</tr>
<tr>
<td>15-4</td>
<td>MAJOR</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SUB, bits [7:4]**

Subtype. Reads as 0x1, to indicate this is a component within a PE.

**MAJOR, bits [3:0]**

Major type. Reads as 0x6, to indicate this is a performance monitor component.

**Accessing the AMDEVTYPE**

AMDEVTYPE can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFCC</td>
<td>AMDEVTYPE</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 15

The AMEVCNTR0<n> characteristics are:

**Purpose**

Provides access to the architected activity monitor event counters.

**Configuration**

External register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR0<n>_EL0[63:0].

External register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch32 System register AMEVCNTR0<n>[63:0].

The power domain of AMEVCNTR0<n> is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n> are RES0.

**Attributes**

AMEVCNTR0<n> is a 64-bit register.

**Field descriptions**

The AMEVCNTR0<n> bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>ACNT</td>
<td>ACNT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**ACNT, bits [63:0]**

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

**Accessing the AMEVCNTR0<n>**

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n> are CONSTRAINED UNPREDICTABLE, and accesses to the register behave as RAZ/WI.

**Note**

AMECGCR.CG0NC identifies the number of architected activity monitor event counters.
AMEVCNTR0\(<n>\) can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0x000 + 8n</td>
<td>AMEVCNTR0(&lt;n&gt;)</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0x004 + 8n</td>
<td>AMEVCNTR0(&lt;n&gt;)</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
AMEVCNTR1<n>, Activity Monitors Event Counter
Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

**Purpose**

Provides access to the auxiliary activity monitor event counters.

**Configuration**

External register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register AMEVCNTR1<n>_EL0[63:0].

External register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch32 System register AMEVCNTR1<n>[63:0].

The power domain of AMEVCNTR1<n> is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are RES0.

**Attributes**

AMEVCNTR1<n> is a 64-bit register.

**Field descriptions**

The AMEVCNTR1<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | ACNT |

**ACNT, bits [63:0]**

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

On a Cold reset, this field resets to 0.

**Accessing the AMEVCNTR1<n>**

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n> are CONSTRAINED UNPREDICTABLE, and accesses to the register behave as RAZ/WI.

**Note**

AMECGCR.CG1NC identifies the number of auxiliary activity monitor event counters.
AMEVCNTR1<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0x100 + 8n</td>
<td>AMEVCNTR1&lt;n&gt;</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0x104 + 8n</td>
<td>AMEVCNTR1&lt;n&gt;</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The AMEVTYPER0<n> characteristics are:

**Purpose**

Provides information on the events that an architected activity monitor event counter AMEVCNTR0<n> counts.

**Configuration**

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPER0<n>_EL0[31:0].

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPER0<n>[31:0].

The power domain of AMEVTYPER0<n> is IMPLEMENTATION DEFINED.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are RES0.

**Attributes**

AMEVTYPER0<n> is a 32-bit register.

**Field descriptions**

The AMEVTYPER0<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RAZ</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>evtCount</td>
</tr>
</tbody>
</table>

**Bits [31:25]**

Reserved, RAZ.

**Bits [24:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

Event to count. The event number of the event that is counted by the architected activity monitor event counter AMEVCNTR0<n>. The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

<table>
<thead>
<tr>
<th>evtCount</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0011</td>
<td>Processor frequency cycles</td>
<td>When n == 0</td>
</tr>
<tr>
<td>0x4004</td>
<td>Constant frequency cycles</td>
<td>When n == 1</td>
</tr>
<tr>
<td>0x0008</td>
<td>Instructions retired</td>
<td>When n == 2</td>
</tr>
<tr>
<td>0x4005</td>
<td>Memory stall cycles</td>
<td>When n == 3</td>
</tr>
</tbody>
</table>
Accessing the AMEVTYPO0\langle n\rangle

If \langle n\rangle is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPO0\langle n\rangle are CONSTRAINED UNPREDICTABLE, and accesses to the register behave as RAZ/WI.

Note

AMCGCR.CG0NC identifies the number of architected activity monitor event counters.

AMEVTYPO0\langle n\rangle can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0x400 + 4n</td>
<td>AMEVTYPO0\langle n\rangle</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The AMEVTYPE1\textsubscript{n} characteristics are:

**Purpose**

Provides information on the events that an auxiliary activity monitor event counter AMEVCNTR1\textsubscript{n} counts.

**Configuration**

External register AMEVTYPE1\textsubscript{n} bits [31:0] are architecturally mapped to AArch64 System register AMEVTYPE1\textsubscript{n}_EL0[31:0].

External register AMEVTYPE1\textsubscript{n} bits [31:0] are architecturally mapped to AArch32 System register AMEVTYPE1\textsubscript{n}[31:0].

The power domain of AMEVTYPE1\textsubscript{n} is IMPLEMENTATION DEFINED.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPE1\textsubscript{n} are RES0.

**Attributes**

AMEVTYPE1\textsubscript{n} is a 32-bit register.

**Field descriptions**

The AMEVTYPE1\textsubscript{n} bit assignments are:

\[
\begin{array}{cccccccccccccccccc}
\hline
\text{RAZ} & \text{RES0} & \text{evtCount} \\
\end{array}
\]

**Bits [31:25]**

Reserved, RAZ.

**Bits [24:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter AMEVCNTR1\textsubscript{n}.

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter AMEVCNTR1\textsubscript{n}, then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

**Note**

The event counted by AMEVCNTR1\textsubscript{n} might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.
If the corresponding counter \texttt{AMEVCNTR1\langle n\rangle} is enabled, writes to this register have \textit{UNPREDICTABLE} results.

\section*{Accessing the AMEVTYPER1\langle n\rangle}

If \( <n> \) is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of \texttt{AMEVTYPER1\langle n\rangle} are \textit{CONSTRAINED UNPREDICTABLE}, and accesses to the register behave as RAZ/WI.

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
Component & Offset & Instance \\
\hline
AMU & 0x480 + 4n & AMEVTYPER1\langle n\rangle \\
\hline
\end{tabular}
\end{center}

Note

\texttt{AMCGCR.CG1NC} identifies the number of auxiliary activity monitor event counters.

\texttt{AMEVTYPER1\langle n\rangle} can be accessed through the memory-mapped interfaces:

Accesses on this interface are \textbf{RO}.
AMIIDR, Activity Monitors Implementation Identification Register

The AMIIDR characteristics are:

**Purpose**

Defines the implementer and revisions of the AMU.

**Configuration**

The power domain of AMIIDR is IMPLEMENTATION DEFINED.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMIIDR are RES0.

**Attributes**

AMIIDR is a 32-bit register.

**Field descriptions**

The AMIIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductID</td>
<td>Variant</td>
<td>Revision</td>
<td>Implementer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ProductID, bits [31:20]**

This field is an AMU part identifier.

The value of this field is IMPLEMENTATION DEFINED.

If AMPIDR0 is implemented, AMPIDR0.PART_0 matches bits [27:20] of this field.

If AMPIDR1 is implemented, AMPIDR1.PART_1 matches bits [31:28] of this field.

**Variant, bits [19:16]**

This field distinguishes product variants or major revisions of the product.

The value of this field is IMPLEMENTATION DEFINED.

If AMPIDR2 is implemented, AMPIDR2.REVISION matches AMIIDR.Variant.

**Revision, bits [15:12]**

This field distinguishes minor revisions of the product.

The value of this field is IMPLEMENTATION DEFINED.

If AMPIDR3 is implemented, AMPIDR3.REVAND matches AMIIDR.Revision.

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the AMU.

For an Arm implementation, this field reads as 0x43B.
Bits [11:8] contain the JEP106 continuation code of the implementer.

Bit 7 is RES0

Bits [6:0] contain the JEP106 identity code of the implementer.

If AMPIDR4 is implemented, AMPIDR4.DES_2 matches bits [11:8] of this field.

If AMPIDR2 is implemented, AMPIDR2.DES_1 matches bits [6:4] of this field.

If AMPIDR1 is implemented, AMPIDR1.DES_0 matches bits [3:0] of this field.

**Accessing the AMIIDR**

**AMIIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xE08</td>
<td>AMIIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The AMPIDR0 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMPIDR0 is **IMPLEMENTATION DEFINED**.

Implementation of this register is **OPTIONAL**.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMPIDR0 are **RES0**.

**Attributes**

AMPIDR0 is a 32-bit register.

**Field descriptions**

The AMPIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>7-0</td>
<td><strong>PART_0</strong></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, **RES0**.

**PART_0, bits [7:0]**

Part number, least significant byte.

The value of this field is **IMPLEMENTATION DEFINED**.

**Accessing the AMPIDR0**

AMPIDR0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFE0</td>
<td>AMPIDR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The AMPIDR1 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMPIDR1 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMPIDR1 are RES0.

**Attributes**

AMPIDR1 is a 32-bit register.

**Field descriptions**

The AMPIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>DES_0</td>
<td>Designer, least significant nibble of JEP106 ID code. The value of this field is IMPLEMENTATION DEFINED. For Arm Limited, this field is 0b1011.</td>
</tr>
<tr>
<td>24</td>
<td>PART_1</td>
<td>Part number, most significant nibble. The value of this field is IMPLEMENTATION DEFINED.</td>
</tr>
</tbody>
</table>

**Accessing the AMPIDR1**

AMPIDR1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFE4</td>
<td>AMPIDR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMPIDR2, Activity Monitors Peripheral Identification Register 2

The AMPIDR2 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMPIDR2 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMPIDR2 are RES0.

**Attributes**

AMPIDR2 is a 32-bit register.

**Field descriptions**

The AMPIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**REVISION, bits [7:4]**

Part major revision. Parts can also use this field to extend Part number to 16-bits.

The value of this field is IMPLEMENTATION DEFINED.

**JEDEC, bit [3]**

RAO. Indicates a JEP106 identity code is used.

**DES_1, bits [2:0]**

Designer, most significant bits of JEP106 ID code.

The value of this field is IMPLEMENTATION DEFINED. For Arm Limited, this field is 0b011.
Accessing the AMPIDR2

AMPIDR2 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFE8</td>
<td>AMPIDR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
AMPIDR3, Activity Monitors Peripheral Identification Register 3

The AMPIDR3 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see "About the Peripheral identification scheme" in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMPIDR3 is **IMPLEMENTATION DEFINED**.

Implementation of this register is **OPTIONAL**.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMPIDR3 are **RES0**.

**Attributes**

AMPIDR3 is a 32-bit register.

**Field descriptions**

The AMPIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>REVAND</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CMOD</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, **RES0**.

**REVAND, bits [7:4]**

Part minor revision. Parts using AMPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

The value of this field is **IMPLEMENTATION DEFINED**.

**CMOD, bits [3:0]**

Customer modified. Indicates someone other than the Designer has modified the component.

The value of this field is **IMPLEMENTATION DEFINED**.

**Accessing the AMPIDR3**

AMPIDR3 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFEC</td>
<td>AMPIDR3</td>
</tr>
</tbody>
</table>
Accesses on this interface are RO.
AMPIDR4, Activity Monitors Peripheral Identification Register 4

The AMPIDR4 characteristics are:

**Purpose**

Provides information to identify an activity monitors component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

The power domain of AMPIDR4 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when AMUv1 is implemented. Otherwise, direct accesses to AMPIDR4 are RES0.

**Attributes**

AMPIDR4 is a 32-bit register.

**Field descriptions**

The AMPIDR4 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 | SIZE | DES_2 |

**Bits [31:8]**

Reserved, RES0.

**SIZE, bits [7:4]**

Size of the component. $\log_2$ of the number of 4KB pages from the start of the component to the end of the component ID registers.

This field reads as 0b0000.

**DES_2, bits [3:0]**

Designer. JEP106 continuation code, least significant nibble.

The value of this field is IMPLEMENTATION DEFINED. For Arm Limited, this field is 0b0100.

**Accessing the AMPIDR4**

AMPIDR4 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMU</td>
<td>0xFD0</td>
<td>AMPIDR4</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RO**.
ASICCTL, CTI External Multiplexer Control register

The ASICCTL characteristics are:

**Purpose**

Can be used to provide implementation defined controls for the CTI. For example, the register might be used to control multiplexors for additional implementation defined triggers. The implementation defined controls provided by this register might modify the architecturally defined behavior of the CTI.

**Note**

The architecturally-defined triggers must not be multiplexed.

**Configuration**

It is implementation defined whether ASICCTL is implemented in the Core power domain or in the Debug power domain.

If it is implemented in the Core power domain then it is implementation defined whether it is in the Cold reset domain or the Warm reset domain.

This register must reset to a value that supports the architecturally-defined behavior of the CTI. Changing the value of the register from its reset value causes implementation defined behavior that might differ from the architecturally-defined behavior of the CTI.

Other than the requirements listed in this register description, all aspects of the reset behavior of the ASICCTL are implementation defined.

**Attributes**

ASICCTL is a 32-bit register.

**Field descriptions**

The ASICCTL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Implementation defined</td>
</tr>
</tbody>
</table>

**Accessing the ASICCTL**

ASICCTL can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x144</td>
<td>ASICCTL</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are RO.
- Otherwise accesses to this register are IMPDEF.
CNTACR<n>, Counter-timer Access Control Registers, n = 0 - 7

The CNTACR<n> characteristics are:

**Purpose**

Provides top-level access controls for the elements of a timer frame. CNTACR<n> provides the controls for frame CNTBaseN.

In addition to the CNTACR<n> control:

- CNTNSAR controls whether CNTACR<n> is accessible by Non-secure accesses.
- If frame CNTEL0BaseN is implemented, the CNTEL0ACR in frame CNTBaseN provides additional control of accesses to frame CNTEL0BaseN.

**Configuration**

The power domain of CNTACR<n> is **IMPLEMENTATION DEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Implemented only if the value of CNTTIDR.Frame<n> is 1.

An implementation of the counters might not provide configurable access to some or all of the features. In this case, the associated field in the CNTACR<n> register is:

- RAZ/WI if access is always denied.
- RAO/WI if access is always permitted.

**Attributes**

CNTACR<n> is a 32-bit register.

**Field descriptions**

The CNTACR<n> bit assignments are:

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>RES0</th>
<th>RWPT</th>
<th>RWVT</th>
<th>RVOFF</th>
<th>RFRO</th>
<th>RVCT</th>
<th>RPCT</th>
</tr>
</thead>
</table>

**Bits [31:6]**

Reserved, RES0.

**RWPT, bit [5]**

Read/write access to the EL1 Physical Timer registers CNTP_CVAL, CNTP_TVAL, and CNTP_CTL in frame <n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th>RWPT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access to the EL1 Physical Timer registers in frame &lt;n&gt;. The registers are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read/write access to the EL1 Physical Timer registers in frame &lt;n&gt;.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally **UNKNOWN** value.

**RWVT, bit [4]**

Read/write access to the Virtual Timer register **CNTV_CVAL**, **CNTV_TVAL**, and **CNTV_CTL**, in frame <n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th>RWVT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access to the Virtual Timer registers in frame &lt;n&gt;. The registers are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read/write access to the Virtual Timer registers in frame &lt;n&gt;.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**RVOFF, bit [3]**

Read-only access to **CNTVOFF**, in frame <n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th>RVOFF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access to <strong>CNTVOFF</strong> in frame &lt;n&gt;. The register is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read-only access to <strong>CNTVOFF</strong> in frame &lt;n&gt;.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**RFRQ, bit [2]**

Read-only access to **CNTFRQ**, in frame <n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th>RFRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access to <strong>CNTFRQ</strong> in frame &lt;n&gt;. The register is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read-only access to <strong>CNTFRQ</strong> in frame &lt;n&gt;.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**RVCT, bit [1]**

Read-only access to **CNTVCT**, in frame <n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th>RVCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access to <strong>CNTVCT</strong> in frame &lt;n&gt;. The register is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read-only access to <strong>CNTVCT</strong> in frame &lt;n&gt;.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**RPCT, bit [0]**

Read-only access to **CNTPCT**, in frame <n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th>RPCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access to <strong>CNTPCT</strong> in frame &lt;n&gt;. The register is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Read-only access to <strong>CNTPCT</strong> in frame &lt;n&gt;.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTACR<n>**

In a system that recognizes two Security states:

- **CNTACR<n>** is always accessible by Secure accesses.
- **CNTNSAR NS<n>** determines whether CNTACR<n> is accessible by Non-secure accesses.
**CNTACR<n>, Counter-timer Access Control Registers, n = 0 - 7**

**CNTACR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTRLBase</td>
<td>0x040 + 4n</td>
<td>CNTACR&lt;n&gt;</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
CNTCR, Counter Control Register

The CNTCR characteristics are:

**Purpose**

Enables the counter, controls the counter frequency setting, and controls counter behavior during debug.

**Configuration**

The power domain of CNTCR is **IMPLEMENTATION DEFINED**. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

**Attributes**

CNTCR is a 32-bit register.

**Field descriptions**

The CNTCR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | FCREQ | RES0 | SCEN | HDBG | EN |

**Bits [31:18]**

Reserved, RES0.

**FCREQ, bits [17:8]**

Frequency change request. Indicates the number of the entry in the Frequency modes table to select.

Selecting an unimplemented entry, or an entry that contains 0, has no effect on the counter.

The maximum number of entries in the Frequency modes table is **IMPLEMENTATION DEFINED** up to a maximum of 1004 entries, see ‘The Frequency modes table’ in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile. An implementation is only required to implement an FCREQ field that can hold values from 0 to the highest supported Frequency modes table entry. Any unrequired most-significant bits of FCREQ can be implemented as RES0.

This field resets to 0.

**Bits [7:3]**

Reserved, RES0.

**SCEN, bit [2]**

When ARMv8.4-CNTSC is implemented:

Scale Enable.

<table>
<thead>
<tr>
<th>SCEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Scaling is not enabled. The counter value is incremented by 0x1.0000000 for each counter tick.</td>
</tr>
<tr>
<td>0b1</td>
<td>Scaling is enabled. The counter is incremented by CNTSCR.ScaleVal for each counter tick.</td>
</tr>
</tbody>
</table>
The SCEN bit can only be changed when the counter is disabled, when CNTCR.EN == 0.

If the value of CNTCR.SCEN changes when CNTCR.EN == 1 then:

- The counter value becomes UNKNOWN.
- The counter value remains UNKNOWN on future ticks of the clock.

When the CNTCV register in the CNTControlBase frame of the memory mapped counter module is written to, the accumulated fraction information is reset to zero.

This field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**HDBG, bit [1]**

Halt-on-debug. Controls whether a Halt-on-debug signal halts the system counter:

<table>
<thead>
<tr>
<th>HDBG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>System counter ignores Halt-on-debug.</td>
</tr>
<tr>
<td>0b1</td>
<td>Asserted Halt-on-debug signal halts system counter update.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**EN, bit [0]**

Enables the counter:

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>System counter disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>System counter enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Accessing the CNTCR**

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

**CNTCR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x000</td>
<td>CNTCR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
CNTCV, Counter Count Value register

The CNTCV characteristics are:

**Purpose**

Indicates the current count value.

**Configuration**

The power domain of CNTCV is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTCV is a 64-bit register.

**Field descriptions**

The CNTCV bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**CountValue, bits [63:0]**

Indicates the counter value.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTCV**

<table>
<thead>
<tr>
<th>Frame</th>
<th>Accessibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNTControlBase</td>
<td>RW</td>
</tr>
<tr>
<td>CNTReadBase</td>
<td>RO</td>
</tr>
</tbody>
</table>

A write to CNTCV must be visible in the CNTPCT register of each running processor in a finite time.

For the instance of the register in the CNTControlBase frame:

- In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, and therefore this register instance, is implemented only in the Secure memory map.
- If the counter is enabled, the effect of writing to the register is UNKNOWN.

In an implementation that supports 64-bit atomic memory accesses, this register must be accessible using a 64-bit atomic access.

**CNTCV can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x008</td>
<td>CNTCV</td>
<td>63.0</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTReadBase</td>
<td>0x000</td>
<td>CNTCV</td>
<td>63:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CNTEL0ACR, Counter-timer EL0 Access Control Register

The CNTEL0ACR characteristics are:

Purpose

An implementation of CNTEL0ACR in the frame at CNTBaseN controls whether the \texttt{CNTPCT}, \texttt{CNTVCT}, \texttt{CNTFRO}, EL1 Physical Timer, and Virtual Timer registers are visible in the frame at CNTEL0BaseN.

Configuration

The power domain of CNTEL0ACR is \texttt{IMPLEMENTATION DEFINED}. RW fields in this register reset to architecturally \texttt{UNKNOWN} values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

Implementation of this register is \texttt{OPTIONAL}.

For more information see ‘Power and reset domains for the system level implementation of the Generic Timer’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

Attributes

CNTEL0ACR is a 32-bit register.

Field descriptions

The CNTEL0ACR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>ELOPTEN</td>
<td>ELOVTEN</td>
<td>RES0</td>
<td>ELOVCTEN</td>
<td>ELOPCTEN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [31:10]

Reserved, RES0.

ELOPTEN, bit [9]

Second view read/write access control for the EL1 Physical Timer registers. This bit controls whether the \texttt{CNTP_CVAL}, \texttt{CNTP_TVAL}, and \texttt{CNTP_CTL} registers in the current CNTBaseN frame are also accessible in the corresponding CNTEL0BaseN frame. The possible values of this bit are:

<table>
<thead>
<tr>
<th>ELOPTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No access. Registers are RES0 in the second view.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access permitted. If the registers are accessible in the current frame then they are accessible in the second view.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally \texttt{UNKNOWN} value.

ELOVTEN, bit [8]

Second view read/write access control for the Virtual Timer registers. This bit controls whether the \texttt{CNTV_CVAL}, \texttt{CNTV_TVAL}, and \texttt{CNTV_CTL} registers in the current CNTBaseN frame are also accessible in the corresponding CNTEL0BaseN frame. The possible values of this bit are:
### EL0VTEN, bit [1]

Second view read access control for **CNTVCT** and **CNTFRQ**. The possible values of this bit are:

<table>
<thead>
<tr>
<th>EL0VTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>CNTVCT</strong> is not visible in the second view. If EL0PCTEN is set to 0, <strong>CNTFRQ</strong> is not visible in the second view.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access permitted. If <strong>CNTVCT</strong> and <strong>CNTFRQ</strong> are visible in the current frame then they are visible in the second view.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### EL0PCTEN, bit [0]

Second view read access control for **CNTPCT** and **CNTFRQ**. The possible values of this bit are:

<table>
<thead>
<tr>
<th>EL0PCTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>CNTPCT</strong> is not visible in the second view. If EL0VTEN is set to 0, <strong>CNTFRQ</strong> is not visible in the second view.</td>
</tr>
<tr>
<td>0b1</td>
<td>Access permitted. If <strong>CNTPCT</strong> and <strong>CNTFRQ</strong> are visible in the current frame then they are visible in the second view.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

### Accessing the CNTEL0ACR

CNTEL0ACR can be implemented in any implemented CNTBaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter 11 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

If CNTEL0ACR is not implemented in an implemented CNTBaseN frame:

- The register location in that frame is RAZ/WI.
- If the corresponding CNTEL0BaseN frame is implemented, the registers **CNTFRQ**, **CNTP_CTL**, **CNTP_CVAL**, **CNTP_TVAL**, **CNTPCT**, **CNTV_CTL**, **CNTV_CVAL**, **CNTV_TVAL**, and **CNTVCT** are not visible in that frame.

### CNTEL0ACR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x014</td>
<td>CNTEL0ACR</td>
</tr>
</tbody>
</table>
Accesses on this interface are RW.
The CNTFID0 characteristics are:

**Purpose**

Indicates the base frequency of the system counter.

**Configuration**

The power domain of CNTFID0 is **IMPLEMENTATION DEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The possible frequencies for the system counter are stored in the Frequency modes table as 32-bit words starting with the base frequency, CNTFID0. For more information see 'The Frequency modes table' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The final entry in the Frequency modes table must be followed by a 32-bit word of zero value, to mark the end of the table.

Typically, the Frequency modes table will be in read-only memory. However, a system implementation might use read/write memory for the table, and initialize the table entries as part of its start-up sequence.

If the Frequency modes table is in read/write memory, Arm strongly recommends that the table is not updated once the system is running.

**Attributes**

CNTFID0 is a 32-bit register.

**Field descriptions**

The CNTFID0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Frequency, bits [31:0]**

The base frequency of the system counter, in Hz.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTFID0**

It is **IMPLEMENTATION DEFINED** whether this register is RO or RW

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

**CNTFID0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x020</td>
<td>CNTFID0</td>
</tr>
</tbody>
</table>
Accesses on this interface are RO or RW.
CNTFID<n>, Counter Frequency IDs, n > 0

The CNTFID<n> characteristics are:

**Purpose**

Indicates alternative system counter update frequencies.

**Configuration**

The power domain of CNTFID<n> is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The possible frequencies for the system counter are stored in the Frequency modes table as 32-bit words starting with the base frequency, CNTFID0, see 'The Frequency modes table' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The number of CNTFID<n> registers is IMPLEMENTATION DEFINED, and the only required CNTFID<n> register is CNTFID0.

The final entry in the Frequency modes table must be followed by a 32-bit word of zero value, to mark the end of the table.

Typically, the Frequency modes table will be in read-only memory. However, a system implementation might use read/write memory for the table, and initialize the table entries as part of its start-up sequence.

If the Frequency modes table is in read/write memory, Arm strongly recommends that the table is not updated once the system is running.

**Attributes**

CNTFID<n> is a 32-bit register.

**Field descriptions**

The CNTFID<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Frequency |

**Frequency, bits [31:0]**

A system counter update frequency, in Hz. Must be an exact divisor of the base frequency. Arm strongly recommends that all frequency values in the Frequency modes table are integer power-of-two divisors of the base frequency.

When the system timer is operating at a lower frequency than the base frequency, the increment applied at each counter update is given by:

\[
\text{increment} = \frac{\text{base frequency}}{\text{selected frequency}}
\]

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTFID<n>**

It is IMPLEMENTATION DEFINED whether this register is RO or RW.
In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes these registers, is implemented only in the Secure memory map.

CNTFID<\(n\)> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x020 + 4(n)</td>
<td>CNTFID&lt;(n)&gt;</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO or RW.
CNTFRQ, Counter-timer Frequency

The CNTFRQ characteristics are:

**Purpose**

This register is provided so that software can discover the frequency of the system counter. The instance of the register in the CNTCTLBase frame must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

**Configuration**

The power domain of CNTFRQ is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTFRQ is a 32-bit register.

**Field descriptions**

The CNTFRQ bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Clock frequency**

Bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTFRQ**

CNTFRQ must be implemented as an RW register in the CNTCTLBase frame.

In a system that recognizes two Security states, the instance of the register in the CNTCTLBase frame is only accessible by Secure accesses.

CNTFRQ can be implemented as a RO register in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I2 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTFRQ is accessible in that frame, as a RO register, if the value of CNTACR<n>.RFRQ is 1.
- Otherwise, the CNTFRQ address in that frame is RAZ/WI.
For an implemented CNTEL0BaseN frame:

- CNTFRQ is accessible as a RO register in that frame if both:
  - CNTFRQ is accessible in the corresponding CNTBaseN frame.
  - Either the value of CNTEL0ACR.EL0VCTEN is 1 or the value of CNTEL0ACR.EL0PCTEN is 1.
- Otherwise, the CNTFRQ address in that frame is RAZ/WI.

**CNTFRQ can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x010</td>
<td>CNTFRQ</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x010</td>
<td>CNTFRQ</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTLBase</td>
<td>0x000</td>
<td>CNTFRQ</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CNTID, Counter Identification Register

The CNTID characteristics are:

**Purpose**

Indicates whether counter scaling is implemented.

**Configuration**

The power domain of CNTID is IMPLEMENTATION DEFINED.

This register is present only when ARMv8.4-CNTSC is implemented. Otherwise, direct accesses to CNTID are RES0.

**Attributes**

CNTID is a 32-bit register.

**Field descriptions**

The CNTID bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>CNTSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**CNTSC, bits [3:0]**

Indicates whether Counter Scaling is implemented

<table>
<thead>
<tr>
<th>CNTSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Counter scaling is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Counter scaling is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the CNTID**

In a system that supports Secure and Non-secure memory maps, the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTID can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x1C</td>
<td>CNTID</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CNTNSAR, Counter-timer Non-secure Access Register

The CNTNSAR characteristics are:

**Purpose**

Provides the highest-level control of whether frames CNTBaseN and CNTEL0BaseN are accessible by Non-secure accesses.

**Configuration**

The power domain of CNTNSAR is **IMPLEMENTATION DEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTNSAR is a 32-bit register.

**Field descriptions**

The CNTNSAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>NS7</td>
</tr>
<tr>
<td>29</td>
<td>NS6</td>
</tr>
<tr>
<td>28</td>
<td>NS5</td>
</tr>
<tr>
<td>27</td>
<td>NS4</td>
</tr>
<tr>
<td>26</td>
<td>NS3</td>
</tr>
<tr>
<td>25</td>
<td>NS2</td>
</tr>
<tr>
<td>24</td>
<td>NS1</td>
</tr>
<tr>
<td>23</td>
<td>NS0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**NS<n>, bit [n], for n = 0 to 7**

Non-secure access to frame n. The possible values of this bit are:

<table>
<thead>
<tr>
<th>NS&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Secure access only. Behaves as RES0 to Non-secure accesses.</td>
</tr>
<tr>
<td>01</td>
<td>Secure and Non-secure accesses permitted.</td>
</tr>
</tbody>
</table>

This bit also determines whether, in the CNTCTLBase frame, CNTACR<n> and CNTVOFF<n> are accessible to Non-secure accesses.

If frame CNTBase<n>:

- Is not implemented, then NS<n> is RES0.
- Is not Configurable access, and is accessible only by Secure accesses, then NS<n> is RES0.
- Is not Configurable access, and is accessible by both Secure and Non-secure accesses, then NS<n> is RES1.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTNSAR**

In a system that recognizes two Security states, this register is only accessible by Secure accesses.
CNTNSAR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTLBase</td>
<td>0x004</td>
<td>CNTNSAR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
CNTP_CTL, Counter-timer Physical Timer Control

The CNTP_CTL characteristics are:

**Purpose**

Control register for the EL1 physical timer.

**Configuration**

The power domain of CNTP_CTL is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTP_CTL is a 32-bit register.

**Field descriptions**

The CNTP_CTL bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>
For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

**ENABLE, bit [0]**

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTP_TVAL** continues to count down.

---

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally **UNKNOWN** value.

### Accessing the CNTP_CTL

CNTP_CTL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTP_CTL is accessible in that frame if the value of **CNTACR<n>.RWPT** is 1.
- Otherwise, the CNTP_CTL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_CTL is accessible in that frame if both:
  - CNTP_CTL is accessible in the corresponding CNTBaseN frame:
  - The value of **CNTEL0ACR.EL0PTEN** is 1.
- Otherwise, the CNTP_CTL address in that frame is RAZ/WI.

**CNTP_CTL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x02C</td>
<td>CNTP_CTL</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x02C</td>
<td>CNTP_CTL</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The CNTP_CVAL characteristics are:

**Purpose**

Holds the 64-bit compare value for the EL1 physical timer.

**Configuration**

The power domain of CNTP_CVAL is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTP_CVAL is a 64-bit register.

**Field descriptions**

The CNTP_CVAL bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 
| 
| 
| 
| CompareValue |
| CompareValue |

**CompareValue, bits [63:0]**

Holds the EL1 physical timer CompareValue.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- An interrupt is generated if CNTP_CTL.IMASK is 0.

When CNTP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTP_CVAL**

CNTP_CVAL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.
For an implemented CNTBaseN frame:

- CNTP_CVAL is accessible in that frame if the value of `CNTACR<n>_.RWPT` is 1.
- Otherwise, the CNTP_CVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_CVAL is accessible in that frame if both:
  - CNTP_CVAL is accessible in the corresponding CNTBaseN frame:
  - The value of `CNTEL0ACR_EL0PTEN` is 1.
- Otherwise, the CNTP_CVAL address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTP_CVAL register must be accessible as an atomic 64-bit value.

**CNTP_CVAL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x020</td>
<td>CNTP_CVAL</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x024</td>
<td>CNTP_CVAL</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x020</td>
<td>CNTP_CVAL</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x024</td>
<td>CNTP_CVAL</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
CNTP_TVAL, Counter-timer Physical Timer TimerValue

The CNTP_TVAL characteristics are:

**Purpose**

Holds the timer value for the EL1 physical timer.

**Configuration**

The power domain of CNTP_TVAL is **IMPLEMENTATION DEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTP_TVAL is a 32-bit register.

**Field descriptions**

The CNTP_TVAL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | TimerValue

**TimerValue, bits [31:0]**

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If CNTP_CTL.ENABLE is 0, the value returned is **UNKNOWN**.
- If CNTP_CTL.ENABLE is 1, the value returned is (CompareValue - CNTPCT).

On a write of this register, CompareValue is set to (CNTPCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTP_CTL.ENABLE is 1, the timer condition is met when (CNTPCT - CompareValue) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTP_CTL.ISTATUS is set to 1.
- If CNTP_CTL.IMASK is 0, an interrupt is generated.

When CNTP_CTL.ENABLE is 0, the timer condition is not met, but CNTPCT continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTP_TVAL**

CNTP_TVAL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:
- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTP_TVAL is accessible in that frame if the value of CNTACR<n>.RWPT is 1.
- Otherwise, the CNTP_TVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_TVAL is accessible in that frame if both:
  - CNTP_TVAL is accessible in the corresponding CNTBaseN frame:
  - The value of CNTEL0ACR.ELOPTEN is 1.
- Otherwise, the CNTP_TVAL address in that frame is RAZ/WI.

**CNTP_TVAL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x028</td>
<td>CNTP_TVAL</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x028</td>
<td>CNTP_TVAL</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
CNTPCT, Counter-timer Physical Count

The CNTPCT characteristics are:

**Purpose**

Holds the 64-bit physical count value.

**Configuration**

The power domain of CNTPCT is IMPLEMENTATION DEFINED.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTPCT is a 64-bit register.

**Field descriptions**

The CNTPCT bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Physical count value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Physical count value.

**Accessing the CNTPCT**

CNTPCT can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame, as a RO register.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTPCT is accessible in that frame, as a RO register, if the value of CNTACR<n>.RPCT is 1.
- Otherwise, the CNTPCT address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTPCT is accessible in that frame if both:
  - CNTPCT is accessible in the corresponding CNTBaseN frame.
  - The value of CNTEL0ACR.EL0PCTEN is 1.
- Otherwise, the CNTPCT address in that frame is RAZ/WI.
If the implementation supports 64-bit atomic accesses, then the CNTPCT register must be accessible as an atomic 64-bit value.

**CNTPCT can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x000</td>
<td>CNTPCT</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x004</td>
<td>CNTPCT</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x000</td>
<td>CNTPCT</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x004</td>
<td>CNTPCT</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CNTSCR, Counter Scale Register

The CNTSCR characteristics are:

**Purpose**

Enables the counter, controls the counter frequency setting, and controls counter behavior during debug.

**Configuration**

The power domain of CNTSCR is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

This register is present only when ARMv8.4-CNTSC is implemented. Otherwise, direct accesses to CNTSCR are RES0.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTSCR is a 32-bit register.

**Field descriptions**

The CNTSCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ScaleVal</td>
</tr>
</tbody>
</table>

**ScaleVal, bits [31:0]**

Scale Value

When counter scaling is enabled, ScaleVal is the amount added to the counter value for every counter tick.

Counter tick is defined as one period of the current operating frequency of the Generic counter.

ScaleVal is expressed as an unsigned fixed point number with an 8-bit integer value and a 24-bit fractional value.

CNTSCR.ScaleVal can only be changed when CNTCR.EN == 0. If the value of this field is changed when CNTCR.EN == 1:

- The counter value becomes UNKNOWN.
- The counter value remains UNKNOWN on future ticks of the clock.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTSCR**

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

**CNTSCR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x10</td>
<td>CNTSCR</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RW**.
The CNTSR characteristics are:

**Purpose**

Provides counter frequency status information.

**Configuration**

The power domain of CNTSR is IMPLEMENTATION DEFINED. Some or all RW fields of this register have defined reset values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTSR is a 32-bit register.

**Field descriptions**

The CNTSR bit assignments are:

```
   31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
   FCACK RES0 DBGH RES0
```

**FCACK, bits [31:8]**

Frequency change acknowledge. Indicates the currently selected entry in the Frequency modes table, see 'The Frequency modes table' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

This field resets to 0.

**Bits [7:2]**

Reserved, RES0.

**DBGH, bit [1]**

Indicates whether the counter is halted because the Halt-on-debug signal is asserted:

<table>
<thead>
<tr>
<th>DBGH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Counter is not halted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Counter is halted.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Bit [0]**

Reserved, RES0.
Accessing the CNTSR

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>0x004</td>
<td>CNTSR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CNTTIDR, Counter-timer Timer ID Register

The CNTTIDR characteristics are:

**Purpose**

Indicates the implemented timers in the memory map, and their features. For each value of N from 0 to 7 it indicates whether:

- Frame CNTBaseN is a view of an implemented timer.
- Frame CNTBaseN has a second view, CNTEL0BaseN.
- Frame CNTBaseN has a virtual timer capability.

**Configuration**

The power domain of CNTTIDR is IMPLEMENTATION DEFINED.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTTIDR is a 32-bit register.

**Field descriptions**

The CNTTIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Frame7</td>
</tr>
<tr>
<td>30</td>
<td>Frame6</td>
</tr>
<tr>
<td>29</td>
<td>Frame5</td>
</tr>
<tr>
<td>28</td>
<td>Frame4</td>
</tr>
<tr>
<td>27</td>
<td>Frame3</td>
</tr>
<tr>
<td>26</td>
<td>Frame2</td>
</tr>
<tr>
<td>25</td>
<td>Frame1</td>
</tr>
<tr>
<td>24</td>
<td>Frame0</td>
</tr>
</tbody>
</table>

**Frame<n>, bits [4n+3:4n], for n = 0 to 7**

A 4-bit field indicating the features of frame CNTBase<n>.

Bit[3] of the field is RES0.

Bit[2], the FEL0 subfield, indicates whether frame CNTBase<n> has a second view, CNTEL0Base<n>. The possible values of this bit are:

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Frame&lt;n&gt; does not have a second view. The CNTEL0ACR register in the first view of the frame is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Frame&lt;n&gt; has a second view, CNTEL0Base&lt;n&gt;.</td>
</tr>
</tbody>
</table>

If bit[0] is 0, bit[2] is RES0.

Bit[1], the FVI subfield, indicates whether both:

- Frame CNTBase<n> implements the virtual timer registers CNTV_CVAL, CNTV_TVAL, and CNTV_CTL.
- This CNTCTLBase frame implements the virtual timer offset register CNTVOFF<n>.

The possible values of bit[1] are:

<table>
<thead>
<tr>
<th>Bit[1]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Frame&lt;n&gt; does not have virtual capability. The virtual time and offset registers are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Frame&lt;n&gt; has virtual capability. The virtual time and offset registers are implemented</td>
</tr>
</tbody>
</table>

If bit[0] is 0, bit[1] is RES0.
Bit[0], the FI subfield, indicates whether frame CNTBase\(<n>\) is implemented. The possible values of this bit are:

<table>
<thead>
<tr>
<th>Bit[0]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Frame(&lt;n&gt;) is not implemented. All registers associated with the frame are RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Frame(&lt;n&gt;) is implemented</td>
</tr>
</tbody>
</table>

### Accessing the CNTTIDR

In a system that recognizes two Security states this register is accessible by both Secure and Non-secure accesses.

**CNTTIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTLBase</td>
<td>0x008</td>
<td>CNTTIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The CNTV_CTL characteristics are:

**Purpose**

Control register for the virtual timer.

**Configuration**

The power domain of CNTV_CTL is **IMPLEMENTATION DEFINED**. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTV_CTL is a 32-bit register.

**Field descriptions**

The CNTV_CTL bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**Bits [31:3]**

Reserved, RES0.

**ISTATUS, bit [2]**

The status of the timer. This bit indicates whether the timer condition is met:

<table>
<thead>
<tr>
<th>ISTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer condition is not met.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer condition is met.</td>
</tr>
</tbody>
</table>

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is **UNKNOWN**.

For more information see 'Operation of the CompareValue views of the timers' and 'Operation of the TimerValue views of the timers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, chapter D6.

This bit is read-only.

**IMASK, bit [1]**

Timer interrupt mask bit. Permitted values are:

<table>
<thead>
<tr>
<th>IMASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer interrupt is not masked by the IMASK bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer interrupt is masked by the IMASK bit.</td>
</tr>
</tbody>
</table>
For more information, see the description of the ISTATUS bit.

This field resets to an architecturally **UNKNOWN** value.

### ENABLE, bit [0]

Enables the timer. Permitted values are:

<table>
<thead>
<tr>
<th>ENABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Timer disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Timer enabled.</td>
</tr>
</tbody>
</table>

Setting this bit to 0 disables the timer output signal, but the timer value accessible from **CNTV_TVAL** continues to count down.

**Note**

Disabling the output signal might be a power-saving option.

This field resets to an architecturally **UNKNOWN** value.

## Accessing the CNTV_CTL

**CNTV_CTL** can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- **CNTV_CTL** is accessible in that frame if the value of **CNTACR<n>**.RWVT is 1.
- Otherwise, the **CNTV_CTL** address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- **CNTV_CTL** is accessible in that frame if both:
  - **CNTV_CTL** is accessible in the corresponding CNTBaseN frame.
  - The value of **CNTEL0ACR**.EL0VTEN is 1.
- Otherwise, the **CNTV_CTL** address in that frame is RAZ/WI.

**CNTV_CTL** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x03C</td>
<td>CNTV_CTL</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x03C</td>
<td>CNTV_CTL</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The CNTV_CVAL characteristics are:

**Purpose**

Holds the 64-bit compare value for the virtual timer.

**Configuration**

The power domain of CNTV_CVAL is `IMPLEMENTATION DEFINED`. RW fields in this register reset to architecturally `UNKNOWN` values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTV_CVAL is a 64-bit register.

**Field descriptions**

The CNTV_CVAL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>CompareValue</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**CompareValue, bits [63:0]**

Holds the virtual timer CompareValue.

When `CNTV_CTL`.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- `CNTV_CTL`.ISTATUS is set to 1.
- An interrupt is generated if `CNTV_CTL`.IMASK is 0.

When `CNTV_CTL`.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count.

This field resets to an architecturally `UNKNOWN` value.

**Accessing the CNTV_CVAL**

CNTV_CVAL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.
For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_CVAL is accessible in that frame if the value of CNTACR<n>.RWVT is 1.
- Otherwise, the CNTV_CVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_CVAL is accessible in that frame if both:
  - CNTV_CVAL is accessible in the corresponding CNTBaseN frame
  - The value of CNTEL0ACR.EL0VTEN is 1.
- Otherwise, the CNTV_CVAL address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTV_CVAL register must be accessible as an atomic 64-bit value.

**CNTV_CVAL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x030</td>
<td>CNTV_CVAL</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x034</td>
<td>CNTV_CVAL</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x030</td>
<td>CNTV_CVAL</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x034</td>
<td>CNTV_CVAL</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
**CNTV_TVAL, Counter-timer Virtual Timer TimerValue**

The CNTV_TVAL characteristics are:

**Purpose**

Holds the timer value for the virtual timer.

**Configuration**

The power domain of CNTV_TVAL is IMPLEMENTATION DEFINED. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a reset of the reset domain in which the register is implemented. The register is not affected by a reset of any other reset domain.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTV_TVAL is a 32-bit register.

**Field descriptions**

The CNTV_TVAL bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**TimerValue, bits [31:0]**

The TimerValue view of the virtual timer.

On a read of this register:

- If CNTV_CTL.ENABLE is 0, the value returned is UNKNOWN.
- If CNTV_CTL.ENABLE is 1, the value returned is (CompareValue - CNTVCT).

On a write of this register, CompareValue is set to (CNTVCT + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When CNTV_CTL.ENABLE is 1, the timer condition is met when (CNTVCT - CompareValue) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- CNTV_CTL.ISTATUS is set to 1.
- If CNTV_CTL.IMASK is 0, an interrupt is generated.

When CNTV_CTL.ENABLE is 0, the timer condition is not met, but CNTVCT continues to count, so the TimerValue view appears to continue to count down.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTV_TVAL**

CNTV_TVAL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:
• Whether the CNTBaseN frame has virtual timer capability.
• Whether the corresponding CNTEL0BaseN frame is implemented.
• For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

• CNTV_TVAL is accessible in that frame if the value of CNTACR<n>.RWVT is 1.
• Otherwise, the CNTV_TVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

• CNTV_TVAL is accessible in that frame if both:
  ◦ CNTV_TVAL is accessible in the corresponding CNTBaseN frame:
  ◦ The value of CNTEL0ACR.EL0VTEN is 1.
• Otherwise, the CNTV_TVAL address in that frame is RAZ/WI.

**CNTV_TVAL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x038</td>
<td>CNTV_TVAL</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x038</td>
<td>CNTV_TVAL</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
CNTVCT, Counter-timer Virtual Count

The CNTVCT characteristics are:

**Purpose**

Holds the 64-bit virtual count value.

**Configuration**

The power domain of CNTVCT is IMPLEMENTATION DEFINED.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTVCT is a 64-bit register.

**Field descriptions**

The CNTVCT bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Virtual count value</td>
<td>Virtual count value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

**Bits [63:0]**

Virtual count value.

**Accessing the CNTVCT**

CNTVCT can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame, as a RO register.

'CNTCTRL status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTVCT is accessible in that frame, as a RO register, if the value of CNTACR\textless n\textgreater .RVCT is 1.
- Otherwise, the CNTVCT address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTVCT is accessible in that frame if both:
  - CNTVCT is accessible in the corresponding CNTBaseN frame.
  - The value of CNTEL0ACR.ELOVCTEN is 1.
- Otherwise, the CNTVCT address in that frame is RAZ/WI.
If the implementation supports 64-bit atomic accesses, then the CNTVCT register must be accessible as an atomic 64-bit value.

**CNTVCT can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x008</td>
<td>CNTVCT</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x00C</td>
<td>CNTVCT</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x008</td>
<td>CNTVCT</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>0x00C</td>
<td>CNTVCT</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CNTVOFF characteristics are:

**Purpose**

Holds the 64-bit virtual offset for a CNTBaseN frame that has virtual timer capability. This is the offset between real time and virtual time.

**Configuration**

The power domain of CNTVOFF is IMPLEMENTATION DEFINED.

On a reset of the reset domain in which an RW instance of this register is implemented, RW fields in the register reset to UNKNOWN values. The register is not affected by a reset of any other reset domain. For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTVOFF is a 64-bit register.

**Field descriptions**

The CNTVOFF bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Virtual offset |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Virtual offset |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |

**Bits [63:0]**

Virtual offset.

This field resets to an architecturally UNKNOWN value.

**Accessing the CNTVOFF**

CNTVOFF is implemented, as a RO register, in any implemented CNTBaseN frame that has virtual timer capability.

'CNTCTRLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTVOFF is accessible in that frame, as a RO register, if the value of CNTACR<n>.RVOFF is 1.
- Otherwise, the CNTVOFF address in that frame is RAZ/WI.

**Note**
CNTVOFF is never visible in any CNTEL0BaseN frame. This means that the CNTVOFF address in any implemented CNTEL0BaseN frame is RAZ/WI.

In an implementation that supports 64-bit atomic accesses, a CNTVOFF{<n>} register must be accessible as an atomic 64-bit value.

**CNTVOFF can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x018</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>0x01C</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CNTVOFF<\text{n}> characteristics are:

**Purpose**

Holds the 64-bit virtual offset for frame CNTBase<\text{n}>. This is the offset between real time and virtual time.

**Configuration**

The power domain of CNTVOFF<\text{n}> is **IMPLEMENTATION DEFINED**.

On a reset of the reset domain in which an RW instance of this register is implemented, RW fields in the register reset to **UNKNOWN** values. The register is not affected by a reset of any other reset domain. For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Attributes**

CNTVOFF<\text{n}> is a 64-bit register.

**Field descriptions**

The CNTVOFF<\text{n}> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Virtual offset</td>
</tr>
<tr>
<td>62</td>
<td>Virtual offset</td>
</tr>
<tr>
<td>31</td>
<td>Virtual offset</td>
</tr>
<tr>
<td>30</td>
<td>Virtual offset</td>
</tr>
<tr>
<td>0</td>
<td>Virtual offset</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Virtual offset.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the CNTVOFF<\text{n}>**

In the CNTCTLBBase frame a CNTVOFF<\text{n}> register must be implemented, as a RW register, for each CNTBaseN frame that has virtual timer capability.

---

**Note**

The value of <\text{n}> in an instance of CNTVOFF<\text{n}> specifies the value of N for the associated CNTBaseN frame.

---

In a system that recognizes two Security states, for any CNTVOFF<\text{n}> register in the CNTCTLBBase frame:

- CNTVOFF<\text{n}> is always accessible by Secure accesses.
- CNTNSAR.NS<\text{n}> determines whether CNTVOFF<\text{n}> is accessible by Non-secure accesses.

The register location of any unimplemented CNTVOFF<\text{n}> register in the CNTCTLBBase frame is RAZ/WI.

The CNTVOFF<\text{n}> register is accessible in the CNTBaseN frame using CNTVOFF.

In an implementation that supports 64-bit atomic accesses, then the CNTVOFF<\text{n}> registers must be accessible as atomic 64-bit values.
CNTVOFF<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTLBase</td>
<td>0x080 + 8n</td>
<td>31:0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTLBase</td>
<td>0x084 + 8n</td>
<td>63:32</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
CounterID<n>, Counter ID registers, n = 0 - 11

The CounterID<n> characteristics are:

**Purpose**

IMPLEMENTATION DEFINED identification registers 0 to 11 for the memory-mapped Generic Timer.

**Configuration**

The power domain of CounterID<n> is IMPLEMENTATION DEFINED.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

These registers are implemented independently in each of the implemented Generic Timer memory-mapped frames.

If the implementation of the Counter ID registers requires an architecture version, the value for this version of the Arm Generic Timer is version 0.

The Counter ID registers can be implemented as a set of CoreSight ID registers, comprising Peripheral ID Registers and Component ID Registers. An implementation of these registers for the Generic Timer must use a Component class value of 0xF.

**Attributes**

CounterID<n> is a 32-bit register.

**Field descriptions**

The CounterID<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

**Accessing the CounterID<n>**

These registers must be implemented, as RO registers, in every implemented Generic Timer memory-mapped frame.

For the CNTCTLBase frame, in a system that recognizes two Security states these registers are accessible by both Secure and Non-secure accesses.

For the CNTControlBase frame, in a system that supports Secure and Non-secure memory maps the frame is implemented only in the Secure memory map, meaning these registers are implemented only in the Secure memory map.

For the CNTBaseN frames, 'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' in Chapter I1 of the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile describes the status fields that identify whether a frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.
CounterID<\text{n}> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTControlBase</td>
<td>$0xFD0 + 4n$</td>
<td>CounterID&lt;\text{n}&gt;</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTReadBase</td>
<td>$0xFD0 + 4n$</td>
<td>CounterID&lt;\text{n}&gt;</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTBaseN</td>
<td>$0xFD0 + 4n$</td>
<td>CounterID&lt;\text{n}&gt;</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTEL0BaseN</td>
<td>$0xFD0 + 4n$</td>
<td>CounterID&lt;\text{n}&gt;</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>CNTCTLBase</td>
<td>$0xFD0 + 4n$</td>
<td>CounterID&lt;\text{n}&gt;</td>
</tr>
</tbody>
</table>
The CTIAPPCLEAR characteristics are:

**Purpose**

Clears bits of the Application Trigger register.

**Configuration**

CTIAPPCLEAR is in the Debug power domain.

**Attributes**

CTIAPPCLEAR is a 32-bit register.

### Field descriptions

The CTIAPPCLEAR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**APPCLEAR<x>, bit [x]**

Application trigger <x> disable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the CTIDEVID.NUMCHAN field. Writing to this bit has the following effect:

<table>
<thead>
<tr>
<th>APPCLEAR&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>Clear corresponding bit in CTIAPPTRIG to 0 and clear the corresponding channel event.</td>
</tr>
</tbody>
</table>

If the ECT does not support multicycle channel events, use of CTIAPPCLEAR is deprecated and the debugger must only use CTIAPPULSE.

### Accessing the CTIAPPCLEAR

CTIAPPCLEAR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x018</td>
<td>CTIAPPCLEAR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **WI**.
- When !SoftwareLockStatus() accesses to this register are **WO**.
CTIAPPULSE, CTI Application Pulse register

The CTIAPPULSE characteristics are:

**Purpose**

Causes event pulses to be generated on ECT channels.

**Configuration**

CTIAPPULSE is in the Debug power domain.

**Attributes**

CTIAPPULSE is a 32-bit register.

**Field descriptions**

The CTIAPPULSE bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>APPULSE&lt;31&gt;, no effect.</td>
</tr>
<tr>
<td>30</td>
<td>APPULSE&lt;30&gt;, channel 0 event pulse generated.</td>
</tr>
<tr>
<td>29</td>
<td>APPULSE&lt;29&gt;, channel 1 event pulse generated.</td>
</tr>
<tr>
<td>28</td>
<td>APPULSE&lt;28&gt;, channel 2 event pulse generated.</td>
</tr>
<tr>
<td>27</td>
<td>APPULSE&lt;27&gt;, channel 3 event pulse generated.</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>APPULSE&lt;0&gt;, channel N event pulse generated.</td>
</tr>
</tbody>
</table>

Writing to this bit has the following effect:

<table>
<thead>
<tr>
<th>APPULSE&lt;31:0&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>No effect.</td>
</tr>
<tr>
<td>01</td>
<td>Channel &lt;x&gt; event pulse generated.</td>
</tr>
</tbody>
</table>

**Note**

- The CTIAPPULSE operation does not affect the state of the Application Trigger register, CTIAPPTRIG. If the channel is active, either because of an earlier event or from the application trigger, then the value written to CTIAPPULSE might have no effect.
- Multiple pulse events that occur close together might be merged into a single pulse event.

**Accessing the CTIAPPULSE**

It is CONSTRAINED UNPREDICTABLE whether a write to CTIAPPULSE generates an event on a channel if CTICONTROL.GLBEN is 0.

CTIAPPULSE can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x01C</td>
<td>CTIAPPULSE</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are WI.
- When !SoftwareLockStatus() accesses to this register are WO.
CTIAPPSET, CTI Application Trigger Set register

The CTIAPPSET characteristics are:

**Purpose**
Sets bits of the Application Trigger register.

**Configuration**
CTIAPPSET is in the Debug power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

**Attributes**
CTIAPPSET is a 32-bit register.

**Field descriptions**
The CTIAPPSET bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>APPSET&lt;x&gt;, bit [x]</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**APPSET<x>, bit [x], for x = 0 to 31**
Application trigger <x> enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the **CTIDEVID.NUMCHAN** field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>APPSET&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reading this means the application trigger is inactive. Writing this has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reading this means the application trigger is active. Writing this sets the corresponding bit in CTIAPTPTRIG to 1 and generates a channel event.</td>
</tr>
</tbody>
</table>

If the ECT does not support multicycle channel events, use of CTIAPPSET is deprecated and the debugger must only use **CTIAPPULSE**.

On a External debug reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the CTIAPPSET**

**CTIAPPSET can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x014</td>
<td>CTIAPPSET</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `SoftwareLockStatus()` accesses to this register are **RO**.
- When `!SoftwareLockStatus()` accesses to this register are **RW**.
CTIAUTHSTATUS, CTI Authentication Status register

The CTIAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for CTI.

Configuration

CTIAUTHSTATUS is in the Debug power domain.

This register is OPTIONAL, and is required for CoreSight compliance.

Attributes

CTIAUTHSTATUS is a 32-bit register.

Field descriptions

The CTIAUTHSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>RAZ</td>
</tr>
<tr>
<td>29</td>
<td>NSNID</td>
</tr>
<tr>
<td>28</td>
<td>NSID</td>
</tr>
<tr>
<td>27</td>
<td>Reserved</td>
</tr>
<tr>
<td>26</td>
<td>Reserved</td>
</tr>
<tr>
<td>25</td>
<td>Reserved</td>
</tr>
<tr>
<td>24</td>
<td>Reserved</td>
</tr>
<tr>
<td>23</td>
<td>Reserved</td>
</tr>
<tr>
<td>22</td>
<td>Reserved</td>
</tr>
<tr>
<td>21</td>
<td>Reserved</td>
</tr>
<tr>
<td>20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19</td>
<td>Reserved</td>
</tr>
<tr>
<td>18</td>
<td>Reserved</td>
</tr>
<tr>
<td>17</td>
<td>Reserved</td>
</tr>
<tr>
<td>16</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>Reserved</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>4</td>
<td>Reserved</td>
</tr>
<tr>
<td>3</td>
<td>Reserved</td>
</tr>
<tr>
<td>2</td>
<td>Reserved</td>
</tr>
<tr>
<td>1</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Bits [31:8]

Reserved, RES0.

Bits [7:4]

Reserved, RAZ.

NSNID, bits [3:2]

If EL3 is not implemented and the implemented Security state is Secure state, holds the same value as DBGAUTHSTATUS_EL1.SNID.

Otherwise, holds the same value as DBGAUTHSTATUS_EL1.NSNID.

NSID, bits [1:0]

If EL3 is not implemented and the implemented Security state is Secure state, holds the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise, holds the same value as DBGAUTHSTATUS_EL1.NSID.

Accessing the CTIAUTHSTATUS

CTIAUTHSTATUS can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFB8</td>
<td>CTIAUTHSTATUS</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTICHINSTATUS, CTI Channel In Status register

The CTICHINSTATUS characteristics are:

Purpose

Provides the raw status of the ECT channel inputs to the CTI.

Configuration

CTICHINSTATUS is in the Debug power domain.

Attributes

CTICHINSTATUS is a 32-bit register.

Field descriptions

The CTICHINSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHIN&lt;n&gt;, bit [n]</td>
<td>Input channel &lt;n&gt; status.</td>
</tr>
</tbody>
</table>

Bits [31:N] are RAZ. N is the number of ECT channels implemented as defined by the CTIDEVID.NUMCHAN field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>CHIN&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input channel &lt;n&gt; is inactive.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input channel &lt;n&gt; is active.</td>
</tr>
</tbody>
</table>

If the ECT channels do not support multicycle events then it is IMPLEMENTATION DEFINED whether an input channel can be observed as active.

Accessing the CTICHINSTATUS

CTICHINSTATUS can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x138</td>
<td>CTICHINSTATUS</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CTICHOUTSTATUS characteristics are:

**Purpose**

Provides the status of the ECT channel outputs from the CTI.

**Configuration**

CTICHOUTSTATUS is in the Debug power domain.

**Attributes**

CTICHOUTSTATUS is a 32-bit register.

**Field descriptions**

The CTICHOUTSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>CHOUT&lt;n&gt;, bit [n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>CHOUT&lt;n&gt;, bit [n] for n = 0 to 31</td>
</tr>
</tbody>
</table>

Output channel <n> status.

Bits [31:N] are RAZ. N is the number of ECT channels implemented as defined by the CTIDEVID.NUMCHAN field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>CHOUT&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Output channel &lt;n&gt; is inactive.</td>
</tr>
<tr>
<td>0b1</td>
<td>Output channel &lt;n&gt; is active.</td>
</tr>
</tbody>
</table>

If the ECT channels do not support multicycle events then it is IMPLEMENTATION DEFINED whether an output channel can be observed as active.

**Note**

The value in CTICHOUTSTATUS is after gating by the channel gate. For more information, see CTIGATE.

**Accessing the CTICHOUTSTATUS**

CTICHOUTSTATUS can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x13C</td>
<td>CTICHOUTSTATUS</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTICIDR0, CTI Component Identification Register 0

The CTICIDR0 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTICIDR0 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

This register is required for CoreSight compliance.

**Attributes**

CTICIDR0 is a 32-bit register.

**Field descriptions**

The CTICIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>27-8</td>
<td>PRMBL_0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_0, bits [7:0]**

Preamble. Must read as 0x0D.

**Accessing the CTICIDR0**

CTICIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFFF0</td>
<td>CTICIDR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CTICIDR1, CTI Component Identification Register 1

The CTICIDR1 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTICIDR1 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

This register is required for CoreSight compliance.

**Attributes**

CTICIDR1 is a 32-bit register.

**Field descriptions**

The CTICIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7:4</td>
<td>Component class. Reads as 0x9, debug component.</td>
</tr>
<tr>
<td>3:0</td>
<td>Preamble. RAZ.</td>
</tr>
</tbody>
</table>

**Accessing the CTICIDR1**

**CTICIDR1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFF4</td>
<td>CTICIDR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The CTICIDR2 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTICIDR2 is in the Debug power domain.

Implementation of this register is Optional.

This register is required for CoreSight compliance.

**Attributes**

CTICIDR2 is a 32-bit register.

**Field descriptions**

The CTICIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7-0</td>
<td>Preamble (PRMBL_2)</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_2, bits [7:0]**

Preamble. Must read as 0x05.

**Accessing the CTICIDR2**

CTICIDR2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFF8</td>
<td>CTICIDR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CTICIDR3 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTICIDR3 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

This register is required for CoreSight compliance.

**Attributes**

CTICIDR3 is a 32-bit register.

**Field descriptions**

The CTICIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>PRMBL_3</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_3, bits [7:0]**

Preamble. Must read as 0xB1.

**Accessing the CTICIDR3**

**CTICIDR3 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFFC</td>
<td>CTICIDR3</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CTICLAIMCLR, CTI CLAIM Tag Clear register

The CTICLAIMCLR characteristics are:

**Purpose**

Used by software to read the values of the CLAIM bits, and to clear CLAIM tag bits to 0.

**Configuration**

CTICLAIMCLR is in the Debug power domain. Some or all RW fields of this register have defined reset values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

Implementation of this register is **OPTIONAL**.

**Attributes**

CTICLAIMCLR is a 32-bit register.

**Field descriptions**

The CTICLAIMCLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | CLAIM[<x>], bit [x] |

**CLAIM[<x>], bit [x], for x = 0 to 31**

CLAIM tag clear bit.

For values of x greater than or equal to the IMPLEMENTATION DEFINED number of CLAIM tags, this bit is RAZ/SBZ. Software can rely on these bits reading as zero, and must use a Should-Be-Zero policy on writes. Implementations must ignore writes.

For other values of x, reads return the value of CLAIM[<x>] and the behavior on writes is:

<table>
<thead>
<tr>
<th>CLAIM[&lt;x&gt;]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Indirectly clear CLAIM[&lt;x&gt;] to 0.</td>
</tr>
</tbody>
</table>

A single write to CTICLAIMCLR can clear multiple tags to 0.

An External Debug reset clears the CLAIM tag bits to 0.

**Accessing the CTICLAIMCLR**

**CTICLAIMCLR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFA4</td>
<td>CTICLAIMCLR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.
CTICLAIMSET, CTI CLAIM Tag Set register

The CTICLAIMSET characteristics are:

**Purpose**

Used by software to set CLAIM bits to 1.

**Configuration**

CTICLAIMSET is in the Debug power domain. Some or all RW fields of this register have defined reset values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

Implementation of this register is **OPTIONAL**.

**Attributes**

CTICLAIMSET is a 32-bit register.

**Field descriptions**

The CTICLAIMSET bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>CLAIM[x] Bit Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 - 0</td>
<td>CLAIM[x], for x = 0 to 31</td>
</tr>
</tbody>
</table>

CLAIM[x], bit [x], for x = 0 to 31

CLAIM tag set bit.

For values of x greater than or equal to the **IMPLEMENTATION DEFINED** number of CLAIM tags, this bit is RAZ/SBZ. Software can rely on these bits reading as zero, and must use a Should-Be-Zero policy on writes. Implementations must ignore writes.

For other values of x, the bit is RAO and the behavior on writes is:

<table>
<thead>
<tr>
<th>CLAIM[x]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Indirectly set CLAIM[x] tag to 1.</td>
</tr>
</tbody>
</table>

A single write to CTICLAIMSET can set multiple tags to 1.

An External Debug reset clears the CLAIM tag bits to 0.

**Accessing the CTICLAIMSET**

CTICLAIMSET can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFA0</td>
<td>CTICLAIMSET</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.
The CTICONTROL characteristics are:

**Purpose**

Controls whether the CTI is enabled.

**Configuration**

CTICONTROL is in the Debug power domain. Some or all RW fields of this register have defined reset values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

**Attributes**

CTICONTROL is a 32-bit register.

**Field descriptions**

The CTICONTROL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>GLBEN (bit 0) Enables or disables the CTI mapping functions. Possible values of this field are:</td>
</tr>
<tr>
<td>29</td>
<td>0b0: CTI mapping functions and application trigger disabled.</td>
</tr>
<tr>
<td>28</td>
<td>0b1: CTI mapping functions and application trigger enabled.</td>
</tr>
</tbody>
</table>

When GLBEN is 0, the input channel to output trigger, input trigger to output channel, and application trigger functions are disabled and do not signal new events on either output triggers or output channels. If a previously asserted output trigger has not been acknowledged, it remains asserted after the mapping functions are disabled. All output triggers are disabled by CTI reset.

If the ECT supports multicycle channel events any existing output channel events will be terminated.

On an External debug reset, this field resets to 0.

**Accessing the CTICONTROL**

CTICONTROL can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x000</td>
<td>CTICONTROL</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are RO.
- When !SoftwareLockStatus() accesses to this register are RW.
CTIDEVAFF0, CTI Device Affinity register 0

The CTIDEVAFF0 characteristics are:

**Purpose**

Copy of the low half of the PE **MPIDR_EL1** register that allows a debugger to determine which PE in a multiprocessor system the CTI component relates to.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

In an Armv8.5 compliant implementation the CTI must be CTIv2.

If this register is implemented, then **CTIDEVAFF1** must also be implemented. If the CTI of a PE does not implement the CTI Device Affinity registers, the CTI block of the external debug memory map must be located 64KB above the debug registers in the external debug interface.

**Configuration**

CTIDEVAFF0 is in the Debug power domain.

Implementation of this register is OPTIONAL.

**Attributes**

CTIDEVAFF0 is a 32-bit register.

**Field descriptions**

The CTIDEVAFF0 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MPIDR_EL1 low half, bits [31:0]**

**MPIDR_EL1** low half. Read-only copy of the low half of **MPIDR_EL1**, as seen from the highest implemented Exception level.

**Accessing the CTIDEVAFF0**

**CTIDEVAFF0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFA8</td>
<td>CTIDEVAFF0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTIDEVAFF1, CTI Device Affinity register 1

The CTIDEVAFF1 characteristics are:

**Purpose**

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the CTI component relates to.

If the CTI is CTIv1, this register is **OPTIONAL**. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

In an Armv8.5 compliant implementation the CTI must be CTIv2.

If this register is implemented, then CTIDEVAFF0 must also be implemented. If the CTI of a PE does not implement the CTI Device Affinity registers, the CTI block of the external debug memory map must be located 64KB above the debug registers in the external debug interface.

**Configuration**

CTIDEVAFF1 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

**Attributes**

CTIDEVAFF1 is a 32-bit register.

**Field descriptions**

The CTIDEVAFF1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**MPIDR_EL1 high half, bits [31:0]**

**MPIDR_EL1** high half. Read-only copy of the high half of **MPIDR_EL1**, as seen from the highest implemented Exception level.

**Accessing the CTIDEVAFF1**

**CTIDEVAFF1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFAC</td>
<td>CTIDEVAFF1</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The CTIDEVARCH characteristics are:

**Purpose**

Identifies the programmers' model architecture of the CTI component.

If the CTI is CTIv1, this register is *optional*. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

In an Armv8.5 compliant implementation the CTI must be CTIv2.

If this register is not implemented, CTIDEVAFF0 and CTIDEVAFF1 are also not implemented.

**Configuration**

CTIDEVARCH is in the Debug power domain.

Implementation of this register is *optional*.

**Attributes**

CTIDEVARCH is a 32-bit register.

**Field descriptions**

The CTIDEVARCH bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHITECT</td>
<td>Defines the architecture of the component. For CTI, this is Arm Limited.</td>
<td></td>
</tr>
<tr>
<td>PRESENT</td>
<td>When set to 1, indicates that the DEVARCH is present.</td>
<td></td>
</tr>
<tr>
<td>REVISION</td>
<td>Revision.</td>
<td></td>
</tr>
<tr>
<td>ARCHID</td>
<td>Defines the architecture revision of the component.</td>
<td></td>
</tr>
</tbody>
</table>

**ARCHITECT, bits [31:21]**

Defines the architecture of the component. For CTI, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

**PRESEN T, bit [20]**

When set to 1, indicates that the DEVARCH is present.

This field is 1 in Armv8.

**REVISION, bits [19:16]**

Revision.

Defines the architecture revision of the component.

<table>
<thead>
<tr>
<th>REVISION</th>
<th>Meaning</th>
<th>Applies when</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>First revision.</td>
<td></td>
</tr>
<tr>
<td>0b0001</td>
<td>As 0b0000, and also adds support for CTIDEVCTL, When ARMv8.3-DoPD is implemented</td>
<td></td>
</tr>
</tbody>
</table>

All other values are reserved.
ARCHID, bits [15:0]

Defines this part to be an Armv8 debug component. For architectures defined by Arm this is further subdivided.

For CTI:

- Bits [15:12] are the architecture version, 0x1.
- Bits [11:0] are the architecture part number, 0xA14.

This corresponds to CTI architecture version CTIv2.

Accessing the CTIDEVARCH

CTIDEVARCH can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFBC</td>
<td>CTIDEVARCH</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTIDEVCTL, CTI Device Control register

The CTIDEVCTL characteristics are:

**Purpose**

Provides target-specific device controls

**Configuration**

CTIDEVCTL is in the Debug power domain. Some or all RW fields of this register have defined reset values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

This register is present only when ARMv8.3-DoPD is implemented. Otherwise, direct accesses to CTIDEVCTL are RES0.

**Attributes**

CTIDEVCTL is a 32-bit register.

**Field descriptions**

The CTIDEVCTL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>RCE</td>
<td>Reset Catch Enable.</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:2]**

Reserved, RES0.

**RCE, bit [1]**

Reset Catch Enable.

<table>
<thead>
<tr>
<th>RCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reset Catch debug event disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset Catch debug event enabled.</td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to 0.

**OSUCE, bit [0]**

OS Unlock Catch Enable

<table>
<thead>
<tr>
<th>OSUCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>OS Unlock Catch debug event disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>OS Unlock Catch debug event enabled.</td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to 0.

**Accessing the CTIDEVCTL**

CTIDEVCTL can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
</table>

Page 3318
This interface is accessible as follows:

- When `SoftwareLockStatus()` accesses to this register are **RO**.
- When `!SoftwareLockStatus()` accesses to this register are **RW**.
CTIDEVID, CTI Device ID register 0

The CTIDEVID characteristics are:

**Purpose**

Describes the CTI component to the debugger.

**Configuration**

CTIDEVID is in the Debug power domain.

**Attributes**

CTIDEVID is a 32-bit register.

**Field descriptions**

The CTIDEVID bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0 INOUT RES0 NUMCHAN RES0 NUMTRIG RES0 EXTMUXNUM</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [31:26]**

Reserved, RES0.

**INOUT, bits [25:24]**

Input/output options. Indicates presence of the input gate. If the CTM is not implemented or CTIv2 is not implemented, this field is RAZ.

<table>
<thead>
<tr>
<th>INOUT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td><strong>CTIGATE</strong> does not mask propagation of input events from external channels.</td>
</tr>
<tr>
<td>0b01</td>
<td><strong>CTIGATE</strong> masks propagation of input events from external channels.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Bits [23:22]**

Reserved, RES0.

**NUMCHAN, bits [21:16]**

Number of ECT channels implemented. IMPLEMENTATION DEFINED. For Armv8, valid values are:

<table>
<thead>
<tr>
<th>NUMCHAN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000011</td>
<td>3 channels (0..2) implemented.</td>
</tr>
<tr>
<td>0b000100</td>
<td>4 channels (0..3) implemented.</td>
</tr>
<tr>
<td>0b000101</td>
<td>5 channels (0..4) implemented.</td>
</tr>
<tr>
<td>0b000110</td>
<td>6 channels (0..5) implemented.</td>
</tr>
</tbody>
</table>

and so on up to 0b100000, 32 channels (0..31) implemented.

All other values are reserved.
Bits [15:14]

Reserved, RES0.

NUMTRIG, bits [13:8]

Number of triggers implemented. IMPLEMENTATION DEFINED. This is one more than the index of the largest trigger, rather than the actual number of triggers.

For Armv8, valid values are:

<table>
<thead>
<tr>
<th>NUMTRIG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000011</td>
<td>Up to 3 triggers (0..2) implemented.</td>
</tr>
<tr>
<td>0b001000</td>
<td>Up to 8 triggers (0..7) implemented.</td>
</tr>
<tr>
<td>0b001001</td>
<td>Up to 9 triggers (0..8) implemented.</td>
</tr>
<tr>
<td>0b001010</td>
<td>Up to 10 triggers (0..9) implemented.</td>
</tr>
</tbody>
</table>

and so on up to 0b100000, 32 triggers (0..31) implemented.

All other values are reserved. If the PE contains a Trace extension, this field must be at least 0b001000. There is no guarantee that any of the implemented triggers, including the highest numbered, are connected to any components.

Bits [7:5]

Reserved, RES0.

EXTMUXNUM, bits [4:0]

Number of multiplexors available on triggers. This value is used in conjunction with External Control register, ASICCTL.

Accessing the CTIDEVID

CTIDEVID can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFC8</td>
<td>CTIDEVID</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTIDEVID1, CTI Device ID register 1

The CTIDEVID1 characteristics are:

**Purpose**

Reserved for future information about the CTI component to the debugger.

**Configuration**

CTIDEVID1 is in the Debug power domain.

**Attributes**

CTIDEVID1 is a 32-bit register.

**Field descriptions**

The CTIDEVID1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
</tbody>
</table>

**Accessing the CTIDEVID1**

CTIDEVID1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFC4</td>
<td>CTIDEVID1</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The CTIDEVID2 characteristics are:

**Purpose**

Reserved for future information about the CTI component to the debugger.

**Configuration**

CTIDEVID2 is in the Debug power domain.

**Attributes**

CTIDEVID2 is a 32-bit register.

**Field descriptions**

The CTIDEVID2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RES0.

**Accessing the CTIDEVID2**

CTIDEVID2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFC0</td>
<td>CTIDEVID2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTIDEVTYPE, CTI Device Type register

The CTIDEVTYPE characteristics are:

**Purpose**

Indicates to a debugger that this component is part of a PEs cross-trigger interface.

**Configuration**

CTIDEVTYPE is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

**Attributes**

CTIDEVTYPE is a 32-bit register.

**Field descriptions**

The CTIDEVTYPE bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | SUB | MAJOR |

**Bits [31:8]**

Reserved, RES0.

**SUB, bits [7:4]**

Subtype. Must read as 0x1 to indicate this is a component within a PE.

**MAJOR, bits [3:0]**

Major type. Must read as 0x4 to indicate this is a cross-trigger component.

**Accessing the CTIDEVTYPE**

CTIDEVTYPE can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFCC</td>
<td>CTIDEVTYPE</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
CTIGATE, CTI Channel Gate Enable register

The CTIGATE characteristics are:

**Purpose**

Determines whether events on channels propagate through the CTM to other ECT components, or from the CTM into the CTI.

**Configuration**

CTIGATE is in the Debug power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

**Attributes**

CTIGATE is a 32-bit register.

**Field descriptions**

The CTIGATE bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

<table>
<thead>
<tr>
<th>GATE&lt;x&gt;, bit [x]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel &lt;x&gt; gate enable.</td>
</tr>
</tbody>
</table>

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the CTIDEVID.NUMCHAN field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>GATE&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Disable output and, if CTIDEVID.INOUT == 0b01, input channel &lt;x&gt; propagation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Enable output and, if CTIDEVID.INOUT == 0b01, input channel &lt;x&gt; propagation.</td>
</tr>
</tbody>
</table>

If GATE[x] is set to 0, no new events will be propagated to the ECT, and if the ECT supports multicycle channel events any existing output channel events will be terminated.

On a External debug reset, this field resets to an architecturally UNKNOWN value.

**Accessing the CTIGATE**

**CTIGATE can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x140</td>
<td>CTIGATE</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.
The CTIINEN<n> characteristics are:

**Purpose**

Enables the signaling of an event on output channels when input trigger event n is received by the CTI.

**Configuration**

CTIINEN<n> is in the Debug power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

If input trigger n is not connected, the behavior of CTIINEN<n> is **IMPLEMENTATION DEFINED**.

**Attributes**

CTIINEN<n> is a 32-bit register.

**Field descriptions**

The CTIINEN<n> bit assignments are:

<table>
<thead>
<tr>
<th>INEN&lt;x&gt;, bit [x]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**INEN<x>, bit [x], for x = 0 to 31**

Input trigger <n> to output channel <x> enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the CTIDEVID.NUMCHAN field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>INEN&lt;x&gt;</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input trigger &lt;n&gt; will not generate an event on output channel &lt;x&gt;.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Input trigger &lt;n&gt; will generate an event on output channel &lt;x&gt;.</td>
<td></td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the CTIINEN<n>**

CTIINEN<n> can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x020 + 4n</td>
<td>CTIINEN&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.
CTIINTACK, CTI Output Trigger Acknowledge register

The CTIINTACK characteristics are:

**Purpose**

Can be used to deactivate the output triggers.

**Configuration**

CTIINTACK is in the Debug power domain.

**Attributes**

CTIINTACK is a 32-bit register.

**Field descriptions**

The CTIINTACK bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ACK&lt;31&gt;</td>
</tr>
<tr>
<td>30</td>
<td>ACK&lt;30&gt;</td>
</tr>
<tr>
<td>29</td>
<td>ACK&lt;29&gt;</td>
</tr>
<tr>
<td>28</td>
<td>ACK&lt;28&gt;</td>
</tr>
<tr>
<td>27</td>
<td>ACK&lt;27&gt;</td>
</tr>
<tr>
<td>26</td>
<td>ACK&lt;26&gt;</td>
</tr>
<tr>
<td>25</td>
<td>ACK&lt;25&gt;</td>
</tr>
<tr>
<td>24</td>
<td>ACK&lt;24&gt;</td>
</tr>
<tr>
<td>23</td>
<td>ACK&lt;23&gt;</td>
</tr>
<tr>
<td>22</td>
<td>ACK&lt;22&gt;</td>
</tr>
<tr>
<td>21</td>
<td>ACK&lt;21&gt;</td>
</tr>
<tr>
<td>20</td>
<td>ACK&lt;20&gt;</td>
</tr>
<tr>
<td>19</td>
<td>ACK&lt;19&gt;</td>
</tr>
<tr>
<td>18</td>
<td>ACK&lt;18&gt;</td>
</tr>
<tr>
<td>17</td>
<td>ACK&lt;17&gt;</td>
</tr>
<tr>
<td>16</td>
<td>ACK&lt;16&gt;</td>
</tr>
<tr>
<td>15</td>
<td>ACK&lt;15&gt;</td>
</tr>
<tr>
<td>14</td>
<td>ACK&lt;14&gt;</td>
</tr>
<tr>
<td>13</td>
<td>ACK&lt;13&gt;</td>
</tr>
<tr>
<td>12</td>
<td>ACK&lt;12&gt;</td>
</tr>
<tr>
<td>11</td>
<td>ACK&lt;11&gt;</td>
</tr>
<tr>
<td>10</td>
<td>ACK&lt;10&gt;</td>
</tr>
<tr>
<td>9</td>
<td>ACK&lt;9&gt;</td>
</tr>
<tr>
<td>8</td>
<td>ACK&lt;8&gt;</td>
</tr>
<tr>
<td>7</td>
<td>ACK&lt;7&gt;</td>
</tr>
<tr>
<td>6</td>
<td>ACK&lt;6&gt;</td>
</tr>
<tr>
<td>5</td>
<td>ACK&lt;5&gt;</td>
</tr>
<tr>
<td>4</td>
<td>ACK&lt;4&gt;</td>
</tr>
<tr>
<td>3</td>
<td>ACK&lt;3&gt;</td>
</tr>
<tr>
<td>2</td>
<td>ACK&lt;2&gt;</td>
</tr>
<tr>
<td>1</td>
<td>ACK&lt;1&gt;</td>
</tr>
<tr>
<td>0</td>
<td>ACK&lt;0&gt;</td>
</tr>
</tbody>
</table>

**ACK<n>, bit [n], for n = 0 to 31**

Acknowledge for output trigger <n>.

Bits [31:N] are RAZ/WI. N is the number of CTI triggers implemented as defined by the CTIDEVID NUMTRIG field.

If any of the following is true, writes to ACK<n> are ignored:

- n >= CTIDEVID NUMTRIG, the number of implemented triggers.
- Output trigger n is not active.
- The channel mapping function output, as controlled by CTIOUTEN<n>, is still active.

Otherwise, if any of the following are true, it is IMPLEMENTATION DEFINED whether writes to ACK<n> are ignored:

- Output trigger n is not implemented.
- Output trigger n is not connected.
- Output trigger n is self-acknowledging and does not require software acknowledge.

Otherwise, the behavior on writes to ACK<n> is as follows:

<table>
<thead>
<tr>
<th>ACK&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No effect</td>
</tr>
<tr>
<td>0b1</td>
<td>Deactivate the trigger</td>
</tr>
</tbody>
</table>

**Accessing the CTIINTACK**

A debugger must read CTITRIGOUTSTATUS to confirm that the output trigger has been acknowledged before generating any event that must be ordered after the write to CTIINTACK, such as a write to CTIAPPPULSE to activate another trigger.

**CTIINTACK can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x010</td>
<td>CTIINTACK</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **WI**.
- When !SoftwareLockStatus() accesses to this register are **WO**.
CTIITCTRL, CTI Integration mode Control register

The CTIITCTRL characteristics are:

**Purpose**

Enables the CTI to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

**Configuration**

It is IMPLEMENTATION DEFINED whether CTIITCTRL is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

Implementation of this register is OPTIONAL.

**Attributes**

CTIITCTRL is a 32-bit register.

**Field descriptions**

The CTIITCTRL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>0</td>
<td>IME</td>
<td>Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection. The integration mode behavior is IMPLEMENTATION DEFINED.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Integration mode enabled.</td>
</tr>
</tbody>
</table>

On a Implementation reset, this field resets to 0.

**Accessing the CTIITCTRL**

CTIITCTRL can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xF00</td>
<td>CTIITCTRL</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
CTIITCTRL, CTI Integration mode Control register

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.
CTILAR, CTI Lock Access Register

The CTILAR characteristics are:

**Purpose**

Allows or disallows access to the CTI registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Cross-Trigger Interface registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Cross-Trigger Interface registers. It does not, and cannot, prevent all accidental or malicious damage.

**Configuration**

CTILAR is in the Debug power domain.

If ARMv8.4-Debug is implemented, the Software Lock is not implemented.

Software uses CTILAR to set or clear the lock, and **CTILSR** to check the current status of the lock.

**Attributes**

CTILAR is a 32-bit register.

**Field descriptions**

The CTILAR bit assignments are:

**When the Software Lock is implemented:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**KEY, bits [31:0]**

Lock Access control. Writing the key value \(0xC5ACCE55\) to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component’s registers through a memory-mapped interface.

**Otherwise:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**RES0**

Reserved, RES0.
Accessing the CTILAR

CTILAR can be accessed through a memory-mapped access to the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFB0</td>
<td>CTILAR</td>
</tr>
</tbody>
</table>

Accesses on this interface are WO.
CTILSR, CTI Lock Status Register

The CTILSR characteristics are:

### Purpose

Indicates the current status of the Software Lock for CTI registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Cross-Trigger Interface registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Cross-Trigger Interface registers. It does not, and cannot, prevent all accidental or malicious damage.

### Configuration

CTILSR is in the Debug power domain. Some or all RW fields of this register have defined reset values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

If ARMv8.4-Debug is implemented, the Software Lock is not implemented.

Software uses **CTILAR** to set or clear the lock, and CTILSR to check the current status of the lock.

### Attributes

CTILSR is a 32-bit register.

### Field descriptions

The CTILSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:3]**

Reserved, RES0.

**nTT, bit [2]**

Not thirty-two bit access required. RAZ.

**SLK, bit [1]**

*When the Software Lock is implemented.*

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when the Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when the Software Lock is implemented, possible values of this field are:

<table>
<thead>
<tr>
<th>SLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Lock clear. Writes are permitted to this component’s registers.</td>
</tr>
<tr>
<td>0b1</td>
<td>Lock set. Writes to this component's registers are ignored, and reads have no side effects.</td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to 1.
Otherwise:

Reserved, RAZ.

**SLI, bit [0]**

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ. For memory-mapped accesses, the value of this field is IMPLEMENTATION DEFINED. Permitted values are:

<table>
<thead>
<tr>
<th>SLI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Software Lock not implemented or not memory-mapped access.</td>
</tr>
<tr>
<td>0b1</td>
<td>Software Lock implemented and memory-mapped access.</td>
</tr>
</tbody>
</table>

**Accessing the CTILSR**

CTILSR can be accessed through a memory-mapped access to the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFB4</td>
<td>CTILSR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CTIOUTEN<n> characteristics are:

**Purpose**

Defines which input channels generate output trigger n.

**Configuration**

CTIOUTEN<n> is in the Debug power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on an External debug reset. The register is not affected by a Warm reset and is not affected by a Cold reset.

If output trigger n is not connected, the behavior of CTIOUTEN<n> is **IMPLEMENTATION DEFINED**.

**Attributes**

CTIOUTEN<n> is a 32-bit register.

**Field descriptions**

The CTIOUTEN<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**OUTEN<x>, bit [x], for x = 0 to 31**

Input channel <x> to output trigger <n> enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the CTIDEVID.NUMCHAN field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>OUTEN&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An event on input channel &lt;x&gt; will not cause output trigger &lt;n&gt; to be asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>An event on input channel &lt;x&gt; will cause output trigger &lt;n&gt; to be asserted.</td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the CTIOUTEN<n>**

CTIOUTEN<n> can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x0A0 + 4n</td>
<td>CTIOUTEN&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.
CTIPIDR0, CTI Peripheral Identification Register 0

The CTIPIDR0 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTIPIDR0 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

This register is required for CoreSight compliance.

**Attributes**

CTIPIDR0 is a 32-bit register.

**Field descriptions**

The CTIPIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7-0</td>
<td>Part number, least significant byte, PART_0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PART_0, bits [7:0]**

Part number, least significant byte.

**Accessing the CTIPIDR0**

CTIPIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFE0</td>
<td>CTIPIDR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The CTIPIDR1 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTIPIDR1 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

**Attributes**

CTIPIDR1 is a 32-bit register.

**Field descriptions**

The CTIPIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>DES_0</td>
</tr>
<tr>
<td>29</td>
<td>PART_1</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Designer, least</td>
</tr>
<tr>
<td>26</td>
<td>significant nibble of</td>
</tr>
<tr>
<td>25</td>
<td>JEP106 ID code. For</td>
</tr>
<tr>
<td>24</td>
<td>Arm Limited, this</td>
</tr>
<tr>
<td>23</td>
<td>field is 0b1011.</td>
</tr>
<tr>
<td>22</td>
<td>Part number, most</td>
</tr>
<tr>
<td>21</td>
<td>significant nibble.</td>
</tr>
</tbody>
</table>

**Accessing the CTIPIDR1**

CTIPIDR1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFE4</td>
<td>CTIPIDR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CTIPIDR2 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTIPIDR2 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

**Attributes**

CTIPIDR2 is a 32-bit register.

**Field descriptions**

The CTIPIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>RESERVED, RES0</td>
</tr>
<tr>
<td>7-4</td>
<td>REVISION, bits [7:4]</td>
</tr>
<tr>
<td>3</td>
<td>JEDEC, bit [3]</td>
</tr>
<tr>
<td>2-0</td>
<td>DES_1, bits [2:0]</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**REVISION, bits [7:4]**

Part major revision. Parts can also use this field to extend Part number to 16-bits.

**JEDEC, bit [3]**

RAO. Indicates a JEP106 identity code is used.

**DES_1, bits [2:0]**

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is 0b011.

**Accessing the CTIPIDR2**

CTIPIDR2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFE8</td>
<td>CTIPIDR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
CTIPIDR3, CTI Peripheral Identification Register 3

The CTIPIDR3 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTIPIDR3 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

This register is required for CoreSight compliance.

**Attributes**

CTIPIDR3 is a 32-bit register.

**Field descriptions**

The CTIPIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>27</td>
<td>REVAND</td>
<td>Part minor revision. Parts using CTIPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.</td>
</tr>
<tr>
<td>23</td>
<td>CMOD</td>
<td>Customer modified. Indicates someone other than the Designer has modified the component.</td>
</tr>
</tbody>
</table>

Accessing the CTIPIDR3

**CTIPIDR3 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFEC</td>
<td>CTIPIDR3</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CTIPIDR4, CTI Peripheral Identification Register 4

The CTIPIDR4 characteristics are:

**Purpose**

Provides information to identify a CTI component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

CTIPIDR4 is in the Debug power domain.

Implementation of this register is **OPTIONAL**.

This register is required for CoreSight compliance.

**Attributes**

CTIPIDR4 is a 32-bit register.

**Field descriptions**

The CTIPIDR4 bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>RES0</th>
<th>SIZE</th>
<th>DES_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SIZE, bits [7:4]**

Size of the component. RAZ. Log2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

**DES_2, bits [3:0]**

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is 0b0100.

**Accessing the CTIPIDR4**

CTIPIDR4 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0xFD0</td>
<td>CTIPIDR4</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
CTITRIGINSTATUS, CTI Trigger In Status register

The CTITRIGINSTATUS characteristics are:

**Purpose**

Provides the status of the trigger inputs.

**Configuration**

CTITRIGINSTATUS is in the Debug power domain.

**Attributes**

CTITRIGINSTATUS is a 32-bit register.

**Field descriptions**

The CTITRIGINSTATUS bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**TRIN<n>, bit [n], for n = 0 to 31**

Trigger input <n> status.

Bits [31:N] are RAZ. N is the number of CTI triggers implemented as defined by the CTIDEVID.NUMTRIG field.

Possible values of this bit are:

<table>
<thead>
<tr>
<th>TRIN&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Input trigger n is inactive.</td>
</tr>
<tr>
<td>0b1</td>
<td>Input trigger n is active.</td>
</tr>
</tbody>
</table>

Not implemented and not-connected input triggers are always inactive.

It is IMPLEMENTATION DEFINED whether an input trigger that does not support multicycle events can be observed as active.

**Accessing the CTITRIGINSTATUS**

CTITRIGINSTATUS can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x130</td>
<td>CTITRIGINSTATUS</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The CTITRIGOUTSTATUS characteristics are:

**Purpose**

Provides the raw status of the trigger outputs, after processing by any IMPLEMENTATION DEFINED trigger interface logic. For output triggers that are self-acknowledging, this is only meaningful if the CTI implements multicycle channel events.

**Configuration**

CTITRIGOUTSTATUS is in the Debug power domain.

**Attributes**

CTITRIGOUTSTATUS is a 32-bit register.

**Field descriptions**

The CTITRIGOUTSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>TROUT&lt;n&gt;</td>
<td>Trigger output &lt;n&gt; status.</td>
</tr>
</tbody>
</table>

Bits [31:N] are RAZ. N is the value in CTIDEVID.NUMTRIG.

If n < N, and output trigger <n> is implemented and connected, and either the trigger is not self-acknowledging or the CTI implements multicycle channel events, then permitted values for TROUT<n> are:

<table>
<thead>
<tr>
<th>TROUT&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Output trigger n is inactive.</td>
</tr>
<tr>
<td>0b1</td>
<td>Output trigger n is active.</td>
</tr>
</tbody>
</table>

Otherwise when n < N it is IMPLEMENTATION DEFINED whether TROUT<n> behaves as described here or is RAZ.

**Accessing the CTITRIGOUTSTATUS**

CTITRIGOUTSTATUS can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTI</td>
<td>0x134</td>
<td>CTITRIGOUTSTATUS</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
DBGAUTHSTATUS_EL1, Debug Authentication Status register

The DBGAUTHSTATUS_EL1 characteristics are:

**Purpose**

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

**Configuration**

External register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch64 System register `DBGAUTHSTATUS_EL1[31:0]`.

External register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register `DBGAUTHSTATUS[31:0]`.

It is IMPLEMENTATION DEFINED whether DBGAUTHSTATUS_EL1 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

DBGAUTHSTATUS_EL1 is a 32-bit register.

**Field descriptions**

The DBGAUTHSTATUS_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>SNID</td>
<td>SID</td>
<td>NSNID</td>
<td>NSID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SNID, bits [7:6]**

When ARMv8.4-Debug is implemented:

Secure non-invasive debug.

`ExternalSecureNoninvasiveDebugEnabled() == ExternalSecureInvasiveDebugEnabled()`.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise:

Secure non-invasive debug.
### SNID, bits [5:4]

Secure invasive debug.

<table>
<thead>
<tr>
<th>SNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3_NS is 1.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

### NSNID, bits [3:2]

When ARMv8.4-Debug is implemented:

Non-secure non-invasive debug.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3_NS is 0.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

If the Effective value of SCR\_EL3\_NS is 1, or if EL3 is implemented and EL2 is not implemented, this field reads as 0b11.

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3_NS is 0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

### NSID, bits [1:0]

Non-secure invasive debug.
<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the DBGAUTHSTATUS_EL1**

DBGAUTHSTATUS_EL1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFB8</td>
<td>DBGAUTHSTATUS_EL1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
The DBGBCR<n>_EL1 characteristics are:

**Purpose**

Holds control information for a breakpoint. Forms breakpoint n together with value register DBGBVR<n>_EL1.

**Configuration**

External register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGBCR<n>_EL1[31:0].

External register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBCR<n>[31:0].

DBGBCR<n>_EL1 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

If breakpoint n is not implemented then accesses to this register are:

- RES0 when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess().
- A CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR otherwise.

**Attributes**

DBGBCR<n>_EL1 is a 32-bit register.

**Field descriptions**

The DBGBCR<n>_EL1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | BT | LBN | SSC | HMC | RES0 | BAS | RES0 | PMC | E |

When the E field is zero, all the other fields in the register are ignored.

**Bits [31:24]**

Reserved, RES0.

**BT, bits [23:20]**

Breakpoint Type. Possible values are:
<table>
<thead>
<tr>
<th>BT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Unlinked instruction address match. <strong>DBGVR&lt;n&gt;_EL1</strong> is the address of an instruction.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As 0b0000 but linked to a Context matching breakpoint.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Unlinked Context ID match. When ARMv8.1-VHE is implemented, EL2 is using AArch64, and the Effective value of <strong>HCR_EL2.E2H</strong> is 1, if either the PE is executing at EL0 with <strong>HCR_EL2.TGE</strong> set to 1 or the PE is executing at EL2, then <strong>DBGVR&lt;n&gt;_EL1.ContextID</strong> must match the <strong>CONTEXTIDR_EL2</strong> value. Otherwise, <strong>DBGVR&lt;n&gt;_EL1.ContextID</strong> must match the <strong>CONTEXTIDR_EL1</strong> value.</td>
</tr>
<tr>
<td>0b0011</td>
<td>As 0b0010, with linking enabled.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Unlinked instruction address mismatch. <strong>DBGVR&lt;n&gt;_EL1</strong> is the address of an instruction to be stepped.</td>
</tr>
<tr>
<td>0b0101</td>
<td>As 0b0100, with linking enabled.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Unlinked <strong>CONTEXTIDR_EL1</strong> match. <strong>DBGVR&lt;n&gt;_EL1.ContextID</strong> is a Context ID compared against <strong>CONTEXTIDR_EL1</strong>.</td>
</tr>
<tr>
<td>0b0111</td>
<td>As 0b0110, with linking enabled.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Unlinked VMID match. <strong>DBGVR&lt;n&gt;_EL1.VMID</strong> is a VMID compared against <strong>VTTBR_EL2.VMID</strong>.</td>
</tr>
<tr>
<td>0b1001</td>
<td>As 0b1000, with linking enabled.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Unlinked VMID and Context ID match. <strong>DBGVR&lt;n&gt;_EL1.ContextID</strong> is a Context ID compared against <strong>CONTEXTIDR_EL1</strong>, and <strong>DBGVR&lt;n&gt;_EL1.VMID</strong> is a VMID compared against <strong>VTTBR_EL2.VMID</strong>.</td>
</tr>
<tr>
<td>0b1011</td>
<td>As 0b1010, with linking enabled.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Unlinked <strong>CONTEXTIDR_EL2</strong> match. <strong>DBGVR&lt;n&gt;_EL1.ContextID2</strong> is a Context ID compared against <strong>CONTEXTIDR_EL2</strong>.</td>
</tr>
<tr>
<td>0b1101</td>
<td>As 0b1100, with linking enabled.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Unlinked Full Context ID match. <strong>DBGVR&lt;n&gt;_EL1.ContextID</strong> is compared against <strong>CONTEXTIDR_EL1</strong>, and <strong>DBGVR&lt;n&gt;_EL1.ContextID2</strong> is compared against <strong>CONTEXTIDR_EL2</strong>.</td>
</tr>
<tr>
<td>0b1111</td>
<td>As 0b1110, with linking enabled.</td>
</tr>
</tbody>
</table>

Constraints on breakpoint programming mean some values are reserved under certain conditions.

For more information on the operation of the SSC, HMC, and PMC fields, and on the effect of programming this field to a reserved value, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug) and 'Reserved DBGBCR<n>_EL1.BT values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**LBN**, bits [19:16]

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an **UNKNOWN** value.

This field is ignored when the value of DBGBCR<n>_EL1.E is 0.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**SSC**, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information, including the effect of programming the fields to a reserved set of values, see 'Reserved DBGBCR<n>_EL1.{SSC, HMC, PMC} values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).
For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**HMC, bit [13]**

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see DBGBCR<n>_EL1 SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [12:9]**

Reserved, **RES0**.

**BAS, bits [8:5]**

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state. In an AArch64 only implementation, this field is reserved, **RES1**.

The permitted values depend on the breakpoint type.

For Address match breakpoints in either AArch32 or AArch64 state, the permitted values are:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Match instruction at</th>
<th>Constraint for debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0011</td>
<td>DBGVR&lt;n&gt;_EL1</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1100</td>
<td>DBGVR&lt;n&gt;_EL1 + 2</td>
<td>Use for T32 instructions</td>
</tr>
<tr>
<td>0b1111</td>
<td>DBGVR&lt;n&gt;_EL1</td>
<td>Use for A64 and A32 instructions</td>
</tr>
</tbody>
</table>

All other values are reserved.

For more information, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

<table>
<thead>
<tr>
<th>BAS</th>
<th>Match instruction at</th>
<th>Constraint for debuggers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>-</td>
<td>Use for a match anywhere breakpoint</td>
</tr>
<tr>
<td>0b0011</td>
<td>DBGVR&lt;n&gt;_EL1</td>
<td>Use for stepping T32 instructions</td>
</tr>
<tr>
<td>0b1100</td>
<td>DBGVR&lt;n&gt;_EL1 + 2</td>
<td>Use for stepping T32 instructions</td>
</tr>
<tr>
<td>0b1111</td>
<td>DBGVR&lt;n&gt;_EL1</td>
<td>Use for stepping A64 and A32 instructions</td>
</tr>
</tbody>
</table>

For more information, see 'Using the BAS field in Address Match breakpoints' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

For Context matching breakpoints, this field is **RES1** and ignored.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [4:3]**

Reserved, **RES0**.
PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the \texttt{DBGBCR<n>\_EL1.SSC} description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

E, bit [0]

Enable breakpoint \texttt{DBGBVR<n>\_EL1}. Possible values are:

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
E & Meaning \\
\hline
0b0 & Breakpoint disabled. \\
0b1 & Breakpoint enabled. \\
\hline
\end{tabular}
\end{table}

On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Accessing the \texttt{DBGBCR<n>\_EL1}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Component & Offset & Instance \\
\hline
Debug & 0x408 + 16n & DBGBCR<n>\_EL1 \\
\hline
\end{tabular}
\end{table}

This interface is accessible as follows:

- When \texttt{IsCorePowered()}, \texttt{!DoubleLockStatus()}, \texttt{!OSLockStatus()}, \texttt{AllowExternalDebugAccess()} and \texttt{SoftwareLockStatus()} accesses to this register are \texttt{RO}.
- When \texttt{IsCorePowered()}, \texttt{!DoubleLockStatus()}, \texttt{!OSLockStatus()}, \texttt{AllowExternalDebugAccess()} and \texttt{!SoftwareLockStatus()} accesses to this register are \texttt{RW}.
- Otherwise accesses to this register generate an error response.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
DBGBVR<n>_EL1, Debug Breakpoint Value Registers, n = 0 - 15

The DBGBVR<n>_EL1 characteristics are:

**Purpose**

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint n together with control register DBGBCR<n>_EL1.

**Configuration**

External register DBGBVR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGBVR<n>_EL1[63:0].

External register DBGBVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGBVR<n>[31:0].

External register DBGBVR<n>_EL1 bits [63:32] are architecturally mapped to AArch32 System register DBGBXVR<n>[31:0].

DBGBVR<n>_EL1 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

If breakpoint n is not implemented then accesses to this register are:

- RES0 when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess().
- A CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR otherwise.

**Attributes**

How this register is interpreted depends on the value of DBGBCR<n>_EL1.BT.

- When DBGBCR<n>_EL1.BT is 0b000x, this register holds a virtual address.
- When DBGBCR<n>_EL1.BT is 0b001x, 0b011x, or 0b110x, this register holds a Context ID.
- When DBGBCR<n>_EL1.BT is 0b100x, this register holds a VMID.
- When DBGBCR<n>_EL1.BT is 0b101x, this register holds a VMID and a Context ID.
- When DBGBCR<n>_EL1.BT is 0b111x, this register holds two Context ID values.

For other values of DBGBCR<n>_EL1.BT, this register is RES0.

**Field descriptions**

The DBGBVR<n>_EL1 bit assignments are:

**When DBGBCR<n>_EL1.BT == 0b000x:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|

**RESS[14:4], bits [63:53]**

Reserved, Sign extended. Software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:
• The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.
• The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.

**VA[52:49], bits [52:49]**

When ARMv8.2-LVA is implemented:


On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise:


**VA[48:2], bits [48:2]**

If the address is being matched in an AArch64 stage 1 translation regime:

• This field contains bits[48:2] of the address for comparison.
• When ARMv8.2-LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, VA[52:49] are RESS.

If the address is being matched in an AArch32 stage 1 translation regime, the first 20 bits of this field are **RES0**, and the rest of the field contains bits[31:2] of the address for comparison.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [1:0]**

Reserved, **RES0**.

**When DBGBCR<n>_EL1.BT == 0b001x:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, **RES0**.

**ContextID, bits [31:0]**

Context ID value for comparison.

The value is compared against **CONTEXTIDR_EL2** when ARMv8.1-VHE is implemented, EL2 is using AArch64, **HCR_EL2.E2H** is 1, and either:

• The PE is executing at EL2.
• **HCR_EL2.TGE** is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

Otherwise, the value is compared against **CONTEXTIDR** when the PE is executing at AArch32, or **CONTEXTIDR_EL1** when the PE is executing at AArch64.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
When DBGBCR<n>_EL1.BT == 0b011x:

| Bits [63:32] | Reserved, RES0. |
| ContextID, bits [31:0] | When ARMv8.1-VHE is implemented: |
| Context ID value for comparison against CONTEXTIDR_EL1. |
| On a Cold reset, this field resets to an architecturally UNKNOWN value. |
| Otherwise: |
| Reserved, RES0. |

When DBGBCR<n>_EL1.BT == 0b100x and HaveEL(EL2):

| Bits [63:48] | Reserved, RES0. |
| VMID[15:8], bits [47:40] | When ARMv8.1-VHE is implemented: |
| Extension to VMID[7:0]. See VMID[7:0] for more details. |
| On a Cold reset, this field resets to an architecturally UNKNOWN value. |
| Otherwise: |
| Reserved, RES0. |

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits in the following cases.

- EL2 is using AArch32.
- ARMv8.1-VMID16 is not implemented.

When ARMv8.1-VMID16 is implemented and EL2 is using AArch64, it is IMPLEMENTATION DEFINED whether the VMID is 8 bits or 16 bits.

VMID[15:8] is RES0 if any of the following applies:

- The implementation has an 8-bit VMID.
• VTCR_EL2.VS has a value of 0.
• EL2 is using AArch32.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [31:0]**

Reserved, RES0.

**When DBGBCR<n>_EL1.BT == 0b101x and HaveEL(EL2):**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | VMID[15:8] | VMID[7:0] |

**Bits [63:48]**

Reserved, RES0.

**VMID[15:8], bits [47:40]**

**When ARMv8.1-VMID16 is implemented:**

Extension to VMID[7:0]. See VMID[7:0] for more details.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**VMID[7:0], bits [39:32]**

VMID value for comparison.

The VMID is 8 bits in the following cases.

• EL2 is using AArch32.
• ARMv8.1-VMID16 is not implemented.

When ARMv8.1-VMID16 is implemented and EL2 is using AArch64, it is **IMPLEMENTATION DEFINED** whether the VMID is 8 bits or 16 bits.

VMID[15:8] is RES0 if any of the following applies:

• The implementation has an 8-bit VMID.
• VTCR_EL2.VS has a value of 0.
• EL2 is using AArch32.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**ContextID, bits [31:0]**

Context ID value for comparison against CONTEXTIDR_EL1.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
When DBGBCR<\text{n}_>EL1.BT == 0b110x and HaveEL(EL2):

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContextID2</td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ContextID2, bits [63:32]

When ARMv8.1-VHE is implemented:

- Context ID value for comparison against CONTEXTIDR_EL2.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

- Reserved, RES0.

Bits [31:0]

Reserved, RES0.

When DBGBCR<\text{n}_>EL1.BT == 0b111x and HaveEL(EL2):

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContextID2</td>
<td>ContextID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

ContextID2, bits [63:32]

When ARMv8.1-VHE is implemented:

- Context ID value for comparison against CONTEXTIDR_EL2.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

- Reserved, RES0.

ContextID, bits [31:0]

When ARMv8.1-VHE is implemented:

- Context ID value for comparison against CONTEXTIDR_EL1.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

- Reserved, RES0.

Accessing the DBGBVR<\text{n}_>EL1

---

Note
SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**DBGBVR<n>_EL1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x40&lt;n&gt; + 16n</td>
<td>DBGBVR&lt;n&gt;_EL1</td>
<td>63:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
The DBGCLAIMCLR_EL1 characteristics are:

**Purpose**

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

**Note**

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMSET_EL1 register.

**Configuration**

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register
DBGCLAIMCLR_EL1[31:0].

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register
DBGCLAIMCLR[31:0].

DBGCLAIMCLR_EL1 is in the Core power domain. Some or all RW fields of this register have defined reset values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

An implementation must include eight CLAIM tag bits.

**Attributes**

DBGCLAIMCLR_EL1 is a 32-bit register.

**Field descriptions**

The DBGCLAIMCLR_EL1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RAZ/SBZ | CLAIM |

**Bits [31:8]**

Reserved, RAZ/SBZ. Software can rely on these bits reading as zero, and must use a should-be-zero policy on writes. Implementations must ignore writes.

**CLAIM, bits [7:0]**

Read or clear CLAIM tag bits. Reading this field returns the current value of the CLAIM tag bits.

Writing a 1 to one of these bits clears the corresponding CLAIM tag bit to 0. This is an indirect write to the CLAIM tag bits. A single write operation can clear multiple CLAIM tag bits to 0.

Writing 0 to one of these bits has no effect.

On a Cold reset, this field resets to 0.
Accessing the DBGCLAIMCLR_EL1

DBGCLAIMCLR_EL1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFA4</td>
<td>DBGCLAIMCLR_EL1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()`, `!OSLockStatus()` and `SoftwareLockStatus()` accesses to this register are **RO**.
- When `IsCorePowered()`, `!DoubleLockStatus()`, `!OSLockStatus()` and `!SoftwareLockStatus()` accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
The DBGCLAIMSET_EL1 characteristics are:

**Purpose**

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

**Note**

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the DBGCLAIMCLR_EL1 register.

**Configuration**

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGCLAIMSET_EL1[31:0].

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGCLAIMSET[31:0].

DBGCLAIMSET_EL1 is in the Core power domain.

An implementation must include eight CLAIM tag bits.

**Attributes**

DBGCLAIMSET_EL1 is a 32-bit register.

**Field descriptions**

The DBGCLAIMSET_EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RAZ/SBZ</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td><strong>CLAIM</strong></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RAZ/SBZ. Software can rely on these bits reading as zero, and must use a should-be-zero policy on writes. Implementations must ignore writes.

**CLAIM, bits [7:0]**

Set CLAIM tag bits.

This field is RAO.

Writing a 1 to one of these bits sets the corresponding CLAIM tag bit to 1. This is an indirect write to the CLAIM tag bits. A single write operation can set multiple CLAIM tag bits to 1.

Writing 0 to one of these bits has no effect.
Accessing the DBGCLAIMSET_EL1

DBGCLAIMSET_EL1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFA0</td>
<td>DBGCLAIMSET_EL1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are RO.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are RW.
- Otherwise accesses to this register generate an error response.
The DBGDTRRX_EL0 characteristics are:

**Purpose**

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communications Channel.

**Configuration**

External register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRRX_EL0[31:0].

External register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRRXint[31:0].

DBGDTRRX_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

**Attributes**

DBGDTRRX_EL0 is a 32-bit register.

**Field descriptions**

The DBGDTRRX_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Update DTRRX |

**Bits [31:0]**

Update DTRRX.

 Writes to this register:

- If RXfull is set to 1, set DTRRX to UNKNOWN.
- If RXfull is set to 0, update the value in DTRRX.

After the write, RXfull is set to 1.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull remains unchanged.

For the full behavior of the Debug Communications Channel, see The Debug Communication Channel and Instruction Transfer Register4.

On a Cold reset, this field resets to an architecturally UNKNOWN value.
Accessing the DBGDTRRX_EL0

If EDSCR.ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any operation issued by a DTR access in memory access mode that has not completed execution is CONSTRAINED UNPREDICTABLE, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

DBGDTRRX_EL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x080</td>
<td>DBGDTRRX_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are RO.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are RW.
- Otherwise accesses to this register generate an error response.
The DBGDTRTX_EL0 characteristics are:

**Purpose**

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See DBGDTR_EL0 for additional architectural mappings. It is a component of the Debug Communication Channel.

**Configuration**

External register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch64 System register DBGDTRTX_EL0[31:0].

External register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register DBGDTRTXint[31:0].

DBGDTRTX_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

**Attributes**

DBGDTRTX_EL0 is a 32-bit register.

**Field descriptions**

The DBGDTRTX_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Return DTRTX

**Bits [31:0]**

Return DTRTX.

Reads of this register:

- If TXfull is set to 1, return the last value written to DTRTX.
- If TXfull is set to 0, return an UNKNOWN value.

After the read, TXfull is cleared to 0.

Writes to this register:

- If TXfull is set to 1, set DTRTX to UNKNOWN.
- If TXfull is set to 0, update the value in DTRTX.

After the write, TXfull remains unchanged.

For the full behavior of the Debug Communications Channel, see The Debug Communication Channel and Instruction Transfer Register.

On a Cold reset, this field resets to an architecturally UNKNOWN value.
Accessing the DBGDTRTX_EL0

If `EDSCR.ITE == 0` when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any operation issued by a DTR access in memory access mode that has not completed execution is **CONSTRAINED UNPREDICTABLE**, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an **UNKNOWN** state.

**DBGDTRTX_EL0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x08C</td>
<td>DBGDTRTX_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()`, `!OSLockStatus()` and `SoftwareLockStatus()` accesses to this register are **RO**.
- When `IsCorePowered()`, `!DoubleLockStatus()`, `!OSLockStatus()` and `!SoftwareLockStatus()` accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
DBGWCR\textsubscript{n} EL1, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR\textsubscript{n} EL1 characteristics are:

**Purpose**

Holds control information for a watchpoint. Forms watchpoint \( \text{n} \) together with value register DBGWVR\textsubscript{n} EL1.

**Configuration**

External register DBGWCR\textsubscript{n} EL1 bits [31:0] are architecturally mapped to AArch64 System register DBGWCR\textsubscript{n} EL1[31:0].

External register DBGWCR\textsubscript{n} EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWCR\textsubscript{n}[31:0].

DBGWCR\textsubscript{n} EL1 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

If watchpoint \( \text{n} \) is not implemented then accesses to this register are:

- When IsCorePowered() \&\& !DoubleLockStatus() \&\& !OSLockStatus() \&\& AllowExternalDebugAccess(), RES0.
- Otherwise, a CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR.

**Attributes**

DBGWCR\textsubscript{n} EL1 is a 32-bit register.

**Field descriptions**

The DBGWCR\textsubscript{n} EL1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | MASK | RES0 | WT | LBN | SSC | HMC | BAS | LSC | PAC | E |

When the E field is zero, all the other fields in the register are ignored.

**Bits [31:29]**

Reserved, RES0.

**MASK, bits [28:24]**

Address mask. Only objects up to 2GB can be watched using a single mask.

<table>
<thead>
<tr>
<th>MASK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>No mask.</td>
</tr>
<tr>
<td>0b00001</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

If programmed with a reserved value, a watchpoint must behave as if either:

- MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCR\textsubscript{n} EL1.
- The watchpoint is disabled.
Software must not rely on this property because the behavior of reserved values might change in a future revision of the architecture.

Other values mask the corresponding number of address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [23:21]**

Reserved, **RES0**.

**WT, bit [20]**

Watchpoint type. Possible values are:

<table>
<thead>
<tr>
<th>WT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Unlinked data address match.</td>
</tr>
<tr>
<td>0b1</td>
<td>Linked data address match.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**LBN, bits [19:16]**

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**SSC, bits [15:14]**

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**HMC, bit [13]**

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**BAS, bits [12:5]**

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by `DBGWVR<n>_EL1` is being watched.

<table>
<thead>
<tr>
<th>BAS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxxx111x</td>
<td>Match byte at <code>DBGWVR&lt;n&gt;_EL1</code> + 1</td>
</tr>
<tr>
<td>xxxx1xxx</td>
<td>Match byte at <code>DBGWVR&lt;n&gt;_EL1</code> + 2</td>
</tr>
<tr>
<td>xxxx11xx</td>
<td>Match byte at <code>DBGWVR&lt;n&gt;_EL1</code> + 3</td>
</tr>
<tr>
<td>xxxx1111</td>
<td>Match byte at <code>DBGWVR&lt;n&gt;_EL1</code> + 4</td>
</tr>
</tbody>
</table>

In cases where `DBGWVR<n>_EL1` addresses a double-word:
BAS Description, if DBGWVR<n>_EL1[2] == 0

<table>
<thead>
<tr>
<th>BAS</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xxx1xxxx</td>
<td>Match byte at DBGWVR&lt;n&gt;_EL1 + 4</td>
</tr>
<tr>
<td>xx1xxxx</td>
<td>Match byte at DBGWVR&lt;n&gt;_EL1 + 5</td>
</tr>
<tr>
<td>x1xxxxxx</td>
<td>Match byte at DBGWVR&lt;n&gt;_EL1 + 6</td>
</tr>
<tr>
<td>1xxxxxxx</td>
<td>Match byte at DBGWVR&lt;n&gt;_EL1 + 7</td>
</tr>
</tbody>
</table>


The valid values for BAS are non-zero binary number all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>.BAS values' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section G2 (AArch32 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**LSC, bits [4:3]**

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

<table>
<thead>
<tr>
<th>LSC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00b1</td>
<td>Match instructions that load from a watchpointed address.</td>
</tr>
<tr>
<td>01b0</td>
<td>Match instructions that store to a watchpointed address.</td>
</tr>
<tr>
<td>01b1</td>
<td>Match instructions that load from or store to a watchpointed address.</td>
</tr>
</tbody>
</table>

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**PAC, bits [2:1]**

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**E, bit [0]**

Enable watchpoint n. Possible values are:

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Watchpoint disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Watchpoint enabled.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the DBGWCR<n>_EL1**

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**DBGWCR<n>_EL1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x808 + 16n</td>
<td>DBGWCR&lt;n&gt;_EL1</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When `IsCorePowered()`, `DoubleLockStatus()`, `OSLockStatus()`, `AllowExternalDebugAccess()` and `SoftwareLockStatus()` accesses to this register are **RO**.
- When `IsCorePowered()`, `DoubleLockStatus()`, `OSLockStatus()`, `AllowExternalDebugAccess()` and `SoftwareLockStatus()` accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
The DBGWVR\textsubscript{\textit{n}} \_ EL1 characteristics are:

**Purpose**

Holds a data address value for use in watchpoint matching. Forms watchpoint \textit{n} together with control register DBGWCR\textsubscript{\textit{n}} \_ EL1.

**Configuration**

External register DBGWVR\textsubscript{\textit{n}} \_ EL1 bits [63:0] are architecturally mapped to AArch64 System register DBGWVR\textsubscript{\textit{n}} \_ EL1[63:0].

External register DBGWVR\textsubscript{\textit{n}} \_ EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGWVR\textsubscript{\textit{n}}[31:0].

DBGWVR\textsubscript{\textit{n}} \_ EL1 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

If watchpoint \textit{n} is not implemented then accesses to this register are:

- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess(), RES0.
- Otherwise, a CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR.

**Attributes**

DBGWVR\textsubscript{\textit{n}} \_ EL1 is a 64-bit register.

**Field descriptions**

The DBGWVR\textsubscript{\textit{n}} \_ EL1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Positions</th>
<th>Field Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-11</td>
<td>Reserved, Sign extended. Hardware and software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1. Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:</td>
</tr>
<tr>
<td>23-12</td>
<td>The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.</td>
</tr>
<tr>
<td>32</td>
<td>The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.</td>
</tr>
</tbody>
</table>

**VA[52:49]**, bits [52:49]

When ARMv8.2-LVA is implemented:


On a Cold reset, this field resets to an architecturally UNKNOWN value.
Otherwise:


**VA[48:2], bits [48:2]**

Bits[48:2] of the address value for comparison.

When ARMv8.2-LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, VA[52:49] are RESS.


On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [1:0]**

Reserved, RES0.

**Accessing the DBGWVR<n>_EL1**

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**DBGWVR<n>_EL1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x800 +</td>
<td>DBGWVR&lt;n&gt;_EL1</td>
<td>63:0</td>
</tr>
<tr>
<td></td>
<td>16n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
EDAA32PFR, External Debug AArch32 Processor Feature Register

The EDAA32PFR characteristics are:

**Purpose**

Provides information about implemented PE features.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.4.1.

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDAA32PFR is implemented in the Core power domain or in the Debug power domain.

This register is present only when the implementation only supports execution in AArch32 state. Otherwise, direct accesses to EDAA32PFR are **RES0**.

**Attributes**

EDAA32PFR is a 64-bit register.

**Field descriptions**

The EDAA32PFR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63-16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15-12</td>
<td>EL3, bits.</td>
</tr>
</tbody>
</table>

**Bits [63:16]**

Reserved, RES0.

**EL3, bits [15:12]**

AArch32 EL3 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL3 is not implemented or can be executed in AArch64 state.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL3 can be executed in AArch32 state only.</td>
</tr>
</tbody>
</table>

When the value of **EDPFR.EL3** is non-zero, this field must be 0b0000.

All other values are reserved.

**Note**

**EDPFR.EL1, EL0** indicate whether EL1 and EL0 can only be executed in AArch32 state.
EL2, bits [11:8]

AArch32 EL2 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL2 is not implemented or can be executed in AArch64 state.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL2 can be executed in AArch32 state only.</td>
</tr>
</tbody>
</table>

When the value of EDPFR.EL2 is non-zero, this field must be 0b0000.

All other values are reserved.

---

Note

EDPFR.{EL1, EL0} indicate whether EL1 and EL0 can only be executed in AArch32 state.

---

PMSA, bits [7:4]

Indicates support for a PMSA. Defined values are:

<table>
<thead>
<tr>
<th>PMSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PMSA not supported.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Support for an Armv8-R PMSAv8-32.</td>
</tr>
</tbody>
</table>

All other values are reserved. In Armv8-A, the only permitted value is 0b0000.

VMSA, bits [3:0]

Indicates support for a VMSA. When the PMSA field is nonzero, determines support for a VMSA. When the PMSA field is 0b0000, VMSA is supported. Defined values are:

<table>
<thead>
<tr>
<th>VMSA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>VMSA not supported.</td>
</tr>
</tbody>
</table>

All other values are reserved. In Armv8-A, the only permitted value is 0b0000.

**Accessing the EDAA32PFR**

EDAA32PFR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x060</td>
<td>EDAA32PFR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.
EDACR, External Debug Auxiliary Control Register

The EDACR characteristics are:

**Purpose**

Allows implementations to support IMPLEMENTATION DEFINED controls.

**Configuration**

It is IMPLEMENTATION DEFINED whether EDACR is implemented in the Core power domain or in the Debug power domain. RW fields in this register reset to architecturally UNKNOWN values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

Changing this register from its reset value causes IMPLEMENTATION DEFINED behavior, including possible deviation from the architecturally-defined behavior.

If the EDACR contains any control bits that must be preserved over power down, then these bits must be accessible by the external debug interface when the OS Lock is locked, OSLR_EL1.OSLK == 1, and when the Core is powered off.

**Attributes**

EDACR is a 32-bit register.

**Field descriptions**

The EDACR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| IMPLEMENTATION DEFINED | IMPLEMENTATION DEFINED, bits [31:0] |

IMPLEMENTATION DEFINED.

This field resets to an architecturally UNKNOWN value.

**Accessing the EDACR**

EDACR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x094</td>
<td>EDACR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are RO.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are RW.
- Otherwise accesses to this register are IMPDEF.
EDCIDR0, External Debug Component Identification Register 0

The EDCIDR0 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is IMPLEMENTATION DEFINED whether EDCIDR0 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDCIDR0 is a 32-bit register.

**Field descriptions**

The EDCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_0, bits [7:0]**

Preamble. Must read as 0x0D.

**Accessing the EDCIDR0**

EDCIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFF0</td>
<td>EDCIDR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDCIDR1, External Debug Component Identification Register 1

The EDCIDR1 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDCIDR1 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDCIDR1 is a 32-bit register.

**Field descriptions**

The EDCIDR1 bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | CLASS | PRMBL_1 |

**Bits [31:8]**

Reserved, RES0.

**CLASS, bits [7:4]**

Component class. Reads as 0x9, debug component.

**PRMBL_1, bits [3:0]**

Preamble. RAZ.

**Accessing the EDCIDR1**

EDCIDR1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFF4</td>
<td>EDCIDR1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
• Otherwise accesses to this register generate an error response.
EDCIDR2, External Debug Component Identification Register 2

The EDCIDR2 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDCIDR2 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDCIDR2 is a 32-bit register.

**Field descriptions**

The EDCIDR2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 | PRMBL_2 |

**Bits [31:8]**

Reserved, RES0.

**PRMBL_2, bits [7:0]**

Preamble. Must read as 0x05.

**Accessing the EDCIDR2**

**EDCIDR2 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFF8</td>
<td>EDCIDR2</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
EDCIFDR3, External Debug Component Identification Register 3

The EDCIFDR3 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is IMPLEMENTATION DEFINED whether EDCIFDR3 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDCIFDR3 is a 32-bit register.

**Field descriptions**

The EDCIFDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>PRMBL_3</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_3, bits [7:0]**

Preamble. Must read as 0xB1.

**Accessing the EDCIFDR3**

**EDCIFDR3 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFFC</td>
<td>EDCIFDR3</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDCIDSRS, External Debug Context ID Sample Register

The EDCIDSRS characteristics are:

**Purpose**

Contains the sampled value of the Context ID, captured on reading EDPCSR[31:0].

**Configuration**

EDCIDSRS is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.0-PCSample is implemented and ARMv8.2-PCSample is not implemented. Otherwise, direct accesses to EDCIDSRS are RES0.

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

**Note**

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

**Attributes**

EDCIDSRS is a 32-bit register.

**Field descriptions**

The EDCIDSRS bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**CONTEXTIDR, bits [31:0]**

Context ID. The value of CONTEXTIDR that is associated with the most recent EDPCSR sample. When the most recent EDPCSR sample was generated:

- If EL1 is using AArch64, then the Context ID is sampled from CONTEXTIDR_EL1.
- If EL1 is using AArch32, then the Context ID is sampled from CONTEXTIDR. If EL3 is implemented and is using AArch32, then CONTEXTIDR is a banked register, and EDCIDSRS samples the current banked copy of CONTEXTIDR for the Security state that is associated with the most recent EDPCSR sample.

Because the value written to EDCIDSRS is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether EDCIDSRS is set to the original or new value if EDPCSR samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the EDCIDSRS**

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN' in chapter H7.
EDCIDSR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x0A4</td>
<td>EDCIDSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
EDDEVAFF0, External Debug Device Affinity register 0

The EDDEVAFF0 characteristics are:

**Purpose**

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the external debug component relates to.

**Configuration**

It is implementation defined whether EDDEVAFF0 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

EDDEVAFF0 is a 32-bit register.

**Field descriptions**

The EDDEVAFF0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>MPIDR_EL1 low half</td>
</tr>
</tbody>
</table>

**MPIDR_EL1 low half, bits [31:0]**

MPIDR_EL1 low half. Read-only copy of the low half of MPIDR_EL1, as seen from the highest implemented Exception level.

**Accessing the EDDEVAFF0**

EDDEVAFF0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFA8</td>
<td>EDDEVAFF0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDDEVAFF1, External Debug Device Affinity register 1

The EDDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the external debug component relates to.

Configuration

It is IMPLEMENTATION DEFINED whether EDDEVAFF1 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVAFF1 is a 32-bit register.

Field descriptions

The EDDEVAFF1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

MPIDR_EL1 high half, bits [31:0]

MPIDR_EL1 high half. Read-only copy of the high half of MPIDR_EL1, as seen from the highest implemented Exception level.

Accessing the EDDEVAFF1

EDDEVAFF1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFAC</td>
<td>EDDEVAFF1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDDEVARCH, External Debug Device Architecture register

The EDDEVARCH characteristics are:

**Purpose**

Identifies the programmers' model architecture of the external debug component.

**Configuration**

It is Implementation Defined whether EDDEVARCH is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is Optional.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

EDDEVARCH is a 32-bit register.

**Field descriptions**

The EDDEVARCH bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ARCHITECT | PRESENT | REVISION | ARCHVER | ARCHPART |

**ARCHITECT, bits [31:21]**

Defines the architecture of the component. For debug, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

**PRESENT, bit [20]**

When set to 1, indicates that the DEVARCH is present.

This field is 1 in Armv8.

**REVISION, bits [19:16]**

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

For debug, the revision defined by Armv8-A is 0x0.

All other values are reserved.

**ARCHVER, bits [15:12]**

Defines the architecture version of the component. This is the same value as ID_AA64DFR0_EL1 DebugVer and DBGIDR Version. The defined values of this field are:
<table>
<thead>
<tr>
<th>ARCHVER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0110</td>
<td>Armv8.0 Debug architecture.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv8.0 Debug architecture with Virtualization Host Extensions.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Armv8.2 Debug architecture.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Armv8.4 Debug architecture.</td>
</tr>
</tbody>
</table>

ARMv8.4-Debug adds the functionality indicated by the value 0b1001. ARMv8.2-Debug adds the functionality indicated by the value 0b1000. If ARMv8.1-VHE is not implemented, the only permitted value is 0b0110.

The fields ARCHVER and ARCHPART together form the field ARCHID, so that ARCHVER is ARCHID[15:12].

**ARCHPART, bits [11:0]**

<table>
<thead>
<tr>
<th>ARCHPART</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA15</td>
<td>The part number of the Armv8-A debug component.</td>
</tr>
</tbody>
</table>

The fields ARCHVER and ARCHPART together form the field ARCHID, so that ARCHPART is ARCHID[11:0].

**Accessing the EDDEVARCH**

**EDDEVARCH can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFBC</td>
<td>EDDEVARCH</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDDEVID, External Debug Device ID register 0

The EDDEVID characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

Configuration

It is implementation defined whether EDDEVID is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVID is a 32-bit register.

Field descriptions

The EDDEVID bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | AuxRegs | RES0 | DebugPower | PCSample |

Bits [31:28]

Reserved, RES0.

AuxRegs, bits [27:24]

Indicates support for Auxiliary registers. Permitted values for this field are:

<table>
<thead>
<tr>
<th>AuxRegs</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>None supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Support for External Debug Auxiliary Control Register, <em>EDACR</em>.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Bits [23:8]

Reserved, RES0.

DebugPower, bits [7:4]

From Armv8.3:

Indicates support for the ARMv8.3-DoPD feature. Defined values of this field are:
### DebugPower

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ARMv8.3-DoPD not implemented. Registers in the external debug interface register map are implemented in a mix of the Debug and Core power domains.</td>
</tr>
<tr>
<td>0b0001</td>
<td>ARMv8.3-DoPD implemented. All registers in the external debug interface register map are implemented in the Core power domain.</td>
</tr>
</tbody>
</table>

ARMv8.3-DoPD implements the functionality added by the value 0b0001.

All other values are reserved.

**Otherwise:**

Reserved, RES0.

### PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers. Permitted values of this field are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PC Sample-based Profiling Extension is not implemented in the external debug registers space.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Only EDPCSR and EDCIDSR are implemented. This option is only permitted if EL3 and EL2 are not implemented.</td>
</tr>
<tr>
<td>0b0011</td>
<td>EDPCSR, EDCIDSR, and EDVIDSR are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When ARMv8.2-PCSample is implemented, the only permitted value is 0b0000.

**Note**

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

### Accessing the EDDEVID

**EDDEVID can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFC8</td>
<td>EDDEVID</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDDEVID1, External Debug Device ID register 1

The EDDEVID1 characteristics are:

**Purpose**

Provides extra information for external debuggers about features of the debug implementation.

**Configuration**

It is implementation defined whether EDDEVID1 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

EDDEVID1 is a 32-bit register.

**Field descriptions**

The EDDEVID1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
</tr>
<tr>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>26</td>
<td>27</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>24</td>
<td>25</td>
</tr>
<tr>
<td>23</td>
<td>24</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
</tr>
<tr>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
</tr>
<tr>
<td>17</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>17</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**PCSROffset, bits [3:0]**

This field indicates the offset applied to PC samples returned by reads of EDPCSR. Permitted values of this field in Armv8 are:

<table>
<thead>
<tr>
<th>PCSROffset</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EDPCSR not implemented.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EDPCSR implemented, and samples have no offset applied and do not sample the instruction set state in AArch32 state.</td>
</tr>
</tbody>
</table>

When ARMv8.2-PCSample is implemented, the only permitted value is 0b0000.

**Note**

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of PMDEVID.PCSample.

**Accessing the EDDEVID1**

EDDEVID1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Debug</td>
<td>0xFC4</td>
<td>EDDEVID1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
EDDEVID2, External Debug Device ID register 2

The EDDEVID2 characteristics are:

**Purpose**

Reserved for future descriptions of features of the debug implementation.

**Configuration**

It is IMPLEMENTATION DEFINED whether EDDEVID2 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

EDDEVID2 is a 32-bit register.

**Field descriptions**

The EDDEVID2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0

**Bits [31:0]**

Reserved, RES0.

**Accessing the EDDEVID2**

EDDEVID2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFC0</td>
<td>EDDEVID2</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDDEVTYPE, External Debug Device Type register

The EDDEVTYPE characteristics are:

**Purpose**

Indicates to a debugger that this component is part of a PEs debug logic.

**Configuration**

It is implementation defined whether EDDEVTYPE is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is optional.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

EDDEVTYPE is a 32-bit register.

**Field descriptions**

The EDDEVTYPE bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>SUB</td>
<td>MAJOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SUB, bits [7:4]**

Subtype. Must read as 0x1 to indicate this is a component within a PE.

**MAJOR, bits [3:0]**

Major type. Must read as 0x5 to indicate this is a debug logic component.

**Accessing the EDDEVTYPE**

EDDEVTYPE can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFCC</td>
<td>EDDEVTYPE</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDDFR, External Debug Feature Register

The EDDFR characteristics are:

**Purpose**

Provides top level information about the debug system.

**Note**

Debuggers must use **EDDEVARCH** to determine the Debug architecture version.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDDFR is implemented in the Core power domain or in the Debug power domain.

**Attributes**

EDDFR is a 64-bit register.

**Field descriptions**

The EDDFR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>TraceFilt</td>
</tr>
<tr>
<td>61</td>
<td>RESERVED</td>
</tr>
<tr>
<td>60</td>
<td>RESERVED</td>
</tr>
<tr>
<td>59</td>
<td>RESERVED</td>
</tr>
<tr>
<td>58</td>
<td>RESERVED</td>
</tr>
<tr>
<td>57</td>
<td>RESERVED</td>
</tr>
<tr>
<td>56</td>
<td>RESERVED</td>
</tr>
<tr>
<td>55</td>
<td>RESERVED</td>
</tr>
<tr>
<td>54</td>
<td>RESERVED</td>
</tr>
<tr>
<td>53</td>
<td>RESERVED</td>
</tr>
<tr>
<td>52</td>
<td>RESERVED</td>
</tr>
<tr>
<td>51</td>
<td>RESERVED</td>
</tr>
<tr>
<td>50</td>
<td>RESERVED</td>
</tr>
<tr>
<td>49</td>
<td>RESERVED</td>
</tr>
<tr>
<td>48</td>
<td>RESERVED</td>
</tr>
<tr>
<td>47</td>
<td>RESERVED</td>
</tr>
<tr>
<td>46</td>
<td>RESERVED</td>
</tr>
<tr>
<td>45</td>
<td>RESERVED</td>
</tr>
<tr>
<td>44</td>
<td>RESERVED</td>
</tr>
<tr>
<td>43</td>
<td>RESERVED</td>
</tr>
<tr>
<td>42</td>
<td>RESERVED</td>
</tr>
<tr>
<td>41</td>
<td>RESERVED</td>
</tr>
<tr>
<td>40</td>
<td>RESERVED</td>
</tr>
<tr>
<td>39</td>
<td>RESERVED</td>
</tr>
<tr>
<td>38</td>
<td>RESERVED</td>
</tr>
<tr>
<td>37</td>
<td>RESERVED</td>
</tr>
<tr>
<td>36</td>
<td>RESERVED</td>
</tr>
<tr>
<td>35</td>
<td>RESERVED</td>
</tr>
<tr>
<td>34</td>
<td>RESERVED</td>
</tr>
<tr>
<td>33</td>
<td>RESERVED</td>
</tr>
<tr>
<td>32</td>
<td>RESERVED</td>
</tr>
</tbody>
</table>

**Bits [63:44]**

Reserved, RES0.

**TraceFilt, bits [43:40]**

**From Armv8.4:**

Armv8.4 Self-hosted Trace Extension version. The defined values of this field are:

<table>
<thead>
<tr>
<th>TraceFilt</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Armv8.4 Self-hosted Trace Extension is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Armv8.4 Self-hosted Trace Extension is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Otherwise:**

Reserved, RES0.
**Bits [39:32]**

Reserved, UNKNOWN.

**CTX_CMPs, bits [31:28]**

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.CTX_CMPs.

**Bits [27:24]**

Reserved, RES0.

**WRPs, bits [23:20]**

Number of watchpoints, minus 1. The value of $0b0000$ is reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.WRPs.

**Bits [19:16]**

Reserved, RES0.

**BRPs, bits [15:12]**

Number of breakpoints, minus 1. The value of $0b0000$ is reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.BRPs.

**PMUVer, bits [11:8]**

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the Alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D10.1.4.

Defined values are:

<table>
<thead>
<tr>
<th>PMUVer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0b0000$</td>
<td>Performance Monitors Extension not implemented.</td>
</tr>
<tr>
<td>$0b0001$</td>
<td>Performance Monitors Extension implemented, PMUv3.</td>
</tr>
</tbody>
</table>
| $0b0100$ | PMUv3 for Armv8.1. As $0b0001$, and also includes support for:  
  • Extended 16-bit PMEVTYPE<_{n}EL0.evtCount field.  
  • If EL2 is implemented, the MDCR_EL2.HPMD control bit. |
| $0b0101$ | PMUv3 for Armv8.4. As $0b0100$ and also includes support for the PMMIR register. |
| $0b0110$ | PMUv3 for Armv8.5. As $0b0101$ and also includes support for:  
  • 64-bit event counters.  
  • If EL2 is implemented, the MDCR_EL2.HCCD control bit.  
  • If EL3 is implemented, the MDCR_EL3.SCCD control bit. |
| $0b1111$ | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value in new implementations. |

ARMv8.1-PMU implements the functionality added by the value $0b0100$. 
ARMv8.4-PMU implements the functionality added by the value 0b0101.

ARMv8.5-PMU implements the functionality added by the value 0b0110.

All other values are reserved.

From Armv8.1, the value 0b0001 is not permitted.

From Armv8.4, the value 0b0100 is not permitted.

From Armv8.5, the value 0b0101 is not permitted.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.PMUVer:

**TraceVer, bits [7:4]**

Trace support. Indicates whether System register interface to a PE trace unit is implemented. Defined values are:

<table>
<thead>
<tr>
<th>TraceVer</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PE trace unit System registers not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PE trace unit System registers implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

A value of 0b0000 only indicates that no System register interface to a PE trace unit is implemented. A PE trace unit might nevertheless be implemented without a System register interface.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64DFR0_EL1.TraceVer:

**Bits [3:0]**

Reserved, UNKNOWN.

**Accessing the EDDFR**

**EDDFR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xD28</td>
<td>EDDFR</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are RO.
- Otherwise accesses to this register are IMPDEF.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xD2C</td>
<td>EDDFR</td>
<td>63:32</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are RO.
- Otherwise accesses to this register are IMPDEF.
The EDECCR characteristics are:

**Purpose**

Controls Exception Catch debug events.

**Configuration**

External register EDECCR bits [31:0] are architecturally mapped to AArch64 System register `OSECCR_EL1[31:0]`.

External register EDECCR bits [31:0] are architecturally mapped to AArch32 System register `DBGOSECCR[31:0]`.

EDECCR is in the Core power domain. Some or all RW fields of this register have defined reset values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

**Attributes**

EDECCR is a 32-bit register.

**Field descriptions**

The EDECCR bit assignments are:

**When ARMv8.2-Debug is implemented:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>NSR3</td>
<td>NSR2</td>
<td>NSR1</td>
<td>NSR0</td>
<td>SR3</td>
<td>SR2</td>
<td>SR1</td>
<td>SR0</td>
<td>NSE3</td>
<td>NSE2</td>
<td>NSE1</td>
<td>NSE0</td>
<td>SE3</td>
<td>SE2</td>
<td>SE1</td>
<td>SE0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, `RES0`.

**NSR<n>, bit [n+12], for n = 0 to 3**

Controls Non-secure exception catch on exception return to EL<n> in conjunction with NSE<n>. See the summary of Exception Catch debug event control for information.

If EL3 is not implemented and the PE behaves as if `SCR_EL3`.NS is set to 0, this field is reserved, `RES0`. Otherwise, possible values for this field are:

<table>
<thead>
<tr>
<th>NSR&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the corresponding NSE&lt;n&gt; bit is 0, then Exception Catch debug events are disabled for Non-secure Exception level &lt;n&gt;. If the corresponding NSE&lt;n&gt; bit is 1, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Non-secure Exception level &lt;n&gt;.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the corresponding NSE&lt;n&gt; bit is 0, then Exception Catch debug events are enabled for exception returns to Non-secure Exception level &lt;n&gt;. If the corresponding NSE&lt;n&gt; bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure Exception level &lt;n&gt;.</td>
</tr>
</tbody>
</table>
It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level is permitted to generate an Exception Catch debug event.

A value of the NSR field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the NSR field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for NSR by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to 0.

SR\(n\), bit \([n+8]\), for \(n = 0\) to 3

Controls Secure exception catch on exception return to EL\(n\) in conjunction with SE\(n\). See the summary of Exception Catch debug event control for information.

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 1, this field is reserved, RES0. Otherwise, possible values for this field are:

<table>
<thead>
<tr>
<th>SR(n)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the corresponding SE(n) bit is 0, then Exception Catch debug events are disabled for Secure Exception level (n). If the corresponding SE(n) bit is 1, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Secure Exception level (n).</td>
</tr>
<tr>
<td>0b1</td>
<td>If the corresponding SE(n) bit is 0, then Exception Catch debug events are enabled for exception returns to Secure Exception level (n). If the corresponding SE(n) bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure Exception level (n).</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

NSE\(n\), bit \([n+4]\), for \(n = 0\) to 3

Coarse-grained Non-secure exception catch for EL\(n\). This controls whether Exception Catch debug events are enabled for Non-secure EL\(n\). This also controls:

- The behavior of exception catch on exception entry to EL\(n\).
- The behavior of exception catch on exception return to EL\(n\) in conjunction with NSR\(n\).

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 0, this field is reserved, RES0. Otherwise, possible values for this field are:
### NSE<n>

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0  If the corresponding NSR&lt;n&gt; bit is 0, then Exception Catch debug events are disabled for Non-secure Exception level &lt;n&gt;.</td>
</tr>
<tr>
<td>If the corresponding NSR&lt;n&gt; bit is 1, then Exception Catch debug events are enabled for exception returns to Non-secure Exception level &lt;n&gt;.</td>
</tr>
<tr>
<td>0b1  If the corresponding NSR&lt;n&gt; bit is 0, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Non-secure Exception level &lt;n&gt;.</td>
</tr>
<tr>
<td>If the corresponding NSR&lt;n&gt; bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure Exception level &lt;n&gt;.</td>
</tr>
</tbody>
</table>

A value of the NSE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the NSE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for NSE by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to 0.

### SE<n>, bit [n], for n = 0 to 3

Coarse-grained Secure exception catch for EL<n>. This field controls whether Exception Catch debug events are enabled for Secure EL<n>.

- The behavior of exception catch on exception entry to EL<n>.
- The behavior of exception catch on exception return to EL<n> in conjunction with SR<n>.

If EL3 is not implemented and the PE behaves as if SCR_EL3.NS is set to 1, this field is reserved, RES0. Otherwise, possible values for this field are:

<table>
<thead>
<tr>
<th>SE&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the corresponding SR&lt;n&gt; bit is 0, then Exception Catch debug events are disabled for Secure Exception level &lt;n&gt;.</td>
</tr>
<tr>
<td>If the corresponding SR&lt;n&gt; bit is 1, then Exception Catch debug events are enabled for exception returns to Secure Exception level &lt;n&gt;.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>If the corresponding SR&lt;n&gt; bit is 0, then Exception Catch debug events are enabled for exception entry, reset entry and exception return to Secure Exception level &lt;n&gt;.</td>
</tr>
<tr>
<td>If the corresponding SR&lt;n&gt; bit is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure Exception level &lt;n&gt;.</td>
<td></td>
</tr>
</tbody>
</table>

A value of the SE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the SE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for SE by a read of EDECCR is UNKNOWN.

On a Cold reset, this field resets to 0.

### Otherwise:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|---------------------------------|
| RES0                           | NSE3                           | NSE2                           | NSE1                           | NSE0                           | SE3                             | SE2                             | SE1                             | SE0                             |

**Bits [31:8]**

Reserved, RES0.
**NSE<\text{n}>**, bit \([n+4]\), for \text{n} = 0 to 3

Coarse-grained Non-secure exception catch. If EL3 and EL2 are not implemented and the PE behaves as if \text{SCR}_{EL3}.NS is set to 0, this field is reserved, RES0. Otherwise, possible values for this field are:

<table>
<thead>
<tr>
<th>NSE&lt;\text{n}&gt;</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception Catch debug events are disabled for Non-secure Exception level &lt;n&gt;.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Exception Catch debug events are enabled for Non-secure Exception level &lt;n&gt;.</td>
<td></td>
</tr>
</tbody>
</table>

A value of the NSE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the NSE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for NSE by a read of EDECCR is \text{UNKNOWN}.

**SE<\text{n}>**, bit \([n]\), for \text{n} = 0 to 3

Coarse-grained Secure exception catch. If EL3 is not implemented and the PE behaves as if \text{SCR}_{EL3}.NS is set to 1, this field is reserved, RES0. Otherwise, possible values for this field are:

<table>
<thead>
<tr>
<th>SE&lt;\text{n}&gt;</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exception Catch debug events are disabled for Secure Exception level &lt;n&gt;.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Exception Catch debug events are enabled for Secure Exception level &lt;n&gt;.</td>
<td></td>
</tr>
</tbody>
</table>

A value of the SE field that enables an Exception Catch debug event for an Exception level that is not implemented is reserved. If the SE field is programmed with a reserved value then:

- The PE behaves as if it is programmed with a defined value, other than for a read of EDECCR.
- The value returned for SE by a read of EDECCR is \text{UNKNOWN}.

**Accessing the EDECCR**

**EDECCR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x098</td>
<td>EDECCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are \text{RO}.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are \text{RW}.
- Otherwise accesses to this register generate an error response.
The EDECR characteristics are:

**Purpose**

Controls Halting debug events.

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDECR is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

EDECR is a 32-bit register.

**Field descriptions**

The EDECR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | SS | RCE | OSUCE |

**Bits [31:3]**

Reserved, RES0.

**SS, bit [2]**

Halting step enable. Possible values of this field are:

<table>
<thead>
<tr>
<th>SS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Halting step debug event disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Halting step debug event enabled.</td>
</tr>
</tbody>
</table>

If the value of EDECR.SS is changed when the PE is in Non-debug state, behavior is **CONSTRAINED UNPREDICTABLE** as described in 'Changing the value of EDECR.SS when not in Debug state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

The following resets apply:

- On a Cold reset, this field resets to:
  - If ARMv8.3-DoPD is implemented, 0.
- On a External debug reset, this field resets to:
  - If ARMv8.3-DoPD is not implemented, 0.
RCE, bit [1]

When ARMv8.3-DoPD is not implemented:

Reset Catch Enable.

<table>
<thead>
<tr>
<th>RCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reset Catch debug event disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset Catch debug event enabled.</td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to 0.

Otherwise:

Reserved, RES0.

OSUCE, bit [0]

When ARMv8.3-DoPD is not implemented:

OS Unlock Catch Enable.

<table>
<thead>
<tr>
<th>OSUCE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>OS Unlock Catch debug event disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>OS Unlock Catch debug event enabled.</td>
</tr>
</tbody>
</table>

On a External debug reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Accessing the EDECR

EDECR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x024</td>
<td>EDECR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When (ARMv8.3-DoPD is not implemented or IsCorePowered()) and SoftwareLockStatus() accesses to this register are RO.
- When (ARMv8.3-DoPD is not implemented or IsCorePowered()) and !SoftwareLockStatus() accesses to this register are RW.
- Otherwise access to this register generate an error response.
EDESR, External Debug Event Status Register

The EDESR characteristics are:

**Purpose**

Indicates the status of internally pending Halting debug events.

**Configuration**

EDESR is in the Core power domain. Some or all RW fields of this register have defined reset values. The field descriptions identify when the reset values apply.

**Attributes**

EDESR is a 32-bit register.

**Field descriptions**

The EDESR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | SS   |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RCOSUC |

**Bits [31:3]**

Reserved, RES0.

**SS, bit [2]**

**When ARMv8.3-DoPD is implemented:**

Halting step debug event pending. Possible values of this field are:

<table>
<thead>
<tr>
<th>SS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reading this means that a Halting step debug event is not pending.</td>
</tr>
<tr>
<td></td>
<td>Writing this means no action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reading this means that a Halting step debug event is pending.</td>
</tr>
<tr>
<td></td>
<td>Writing this clears the pending Halting step debug event.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**Otherwise:**

Halting step debug event pending. Possible values of this field are:

<table>
<thead>
<tr>
<th>SS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reading this means that a Halting step debug event is not pending.</td>
</tr>
<tr>
<td></td>
<td>Writing this means no action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reading this means that a Halting step debug event is pending.</td>
</tr>
<tr>
<td></td>
<td>Writing this clears the pending Halting step debug event.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to the value in EDECR.SS.

**RC, bit [1]**

Reset Catch debug event pending. Possible values of this field are:
## RC, bit [0]

Reading this means that a Reset Catch debug event is not pending. Writing this means no action.

Reading this means that a Reset Catch debug event is pending. Writing this clears the pending Reset Catch debug event.

On a Warm reset, this field resets to:

- If ARMv8.3-DoPD is implemented, The value in `CTIDEVCTL.RCE`.
- If ARMv8.3-DoPD is not implemented, The value in `EDECR.RCE`.

### OSUC, bit [0]

OS Unlock Catch debug event pending. Possible values of this field are:

<table>
<thead>
<tr>
<th>OSUC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Reading this means that an OS Unlock Catch debug event is not pending. Writing this means no action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reading this means that an OS Unlock Catch debug event is pending. Writing this clears the pending OS Unlock Catch debug event.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to 0.

## Accessing the EESDR

If a request to clear a pending Halting debug event is received at or about the time when halting becomes allowed, it is constrained unpredictable whether the event is taken.

If Core power is removed while a Halting debug event is pending, it is lost. However, it might become pending again when the Core is powered back on and Cold reset.

### EESDR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x020</td>
<td>EESDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()` and `SoftwareLockStatus()` accesses to this register are RO.
- When `IsCorePowered()`, `!DoubleLockStatus()` and `!SoftwareLockStatus()` accesses to this register are RW.
- Otherwise accesses to this register generate an error response.
EDITCTRL, External Debug Integration mode Control register

The EDITCTRL characteristics are:

**Purpose**

Enables the external debug to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

**Configuration**

It is IMPLEMENTATION DEFINED whether EDITCTRL is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

Implementation of this register is OPTIONAL.

**Attributes**

EDITCTRL is a 32-bit register.

**Field descriptions**

The EDITCTRL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>30</td>
<td>IME</td>
<td>Integration mode enable</td>
</tr>
</tbody>
</table>

**Bits [31:1]**

Reserved, RES0.

**IME, bit [0]**

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection. The integration mode behavior is IMPLEMENTATION DEFINED.

<table>
<thead>
<tr>
<th>IME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Integration mode enabled.</td>
</tr>
</tbody>
</table>

On a Implementation reset, this field resets to 0.

**Accessing the EDITCTRL**

EDITCTRL can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xF00</td>
<td>EDITCTRL</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.
**EDITR, External Debug Instruction Transfer Register**

The EDITR characteristics are:

**Purpose**

Used in Debug state for passing instructions to the PE for execution.

**Configuration**

EDITR is in the Core power domain.

**Attributes**

EDITR is a 32-bit register.

**Field descriptions**

The EDITR bit assignments are:

**When in AArch32 state:**

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

<table>
<thead>
<tr>
<th>31:16</th>
<th>15:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>T32Second</td>
<td>T32First</td>
</tr>
</tbody>
</table>

T32Second, bits [31:16]

Second halfword of the T32 instruction to be executed on the PE. When EDITR contains a 16-bit T32 instruction, this field is ignored. For more information see 'Behavior in Debug state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H2, Debug State.

T32First, bits [15:0]

First halfword of the T32 instruction to be executed on the PE.

**When in AArch64 state:**

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

A64 instruction to be executed on the PE.

**Bits [31:0]**

A64 instruction to be executed on the PE.

**Accessing the EDITR**

If **EDSCR**, ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any instruction issued through the ITR in Normal access mode that has not completed execution is CONSTRAINED UNPREDICTABLE, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.
EDITR ignores writes if the PE is in Non-debug state.

**EDITR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x084</td>
<td>EDITR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **WI**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.
EDLAR, External Debug Lock Access Register

The EDLAR characteristics are:

**Purpose**

Allows or disallows access to the external debug registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the debug registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the debug registers. It does not, and cannot, prevent all accidental or malicious damage.

**Configuration**

It is defined whether EDLAR is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, then ARMv8.0-SoftwareLock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If ARMv8.3-DoPD is not implemented, then this register is in the Debug power domain.

Software uses EDLAR to set or clear the lock, and EDLSR to check the current status of the lock.

**Attributes**

EDLAR is a 32-bit register.

**Field descriptions**

The EDLAR bit assignments are:

**When the Software Lock is implemented:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>KEY</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

KEY, bits [31:0]

Lock Access control. Writing the key value 0xC5ACCE55 to this field unlocks the lock, enabling write accesses to this component’s registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component’s registers through a memory mapped interface.

**Otherwise:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Otherwise

**Bits [31:0]**

Reserved, RES0.
Accessing the EDLAR

EDLAR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFB0</td>
<td>EDLAR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are WO.
- Otherwise accesses to this register generate an error response.
EDLSR, External Debug Lock Status Register

The EDLSR characteristics are:

**Purpose**

Indicates the current status of the software lock for external debug registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the debug registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the debug registers. It does not, and cannot, prevent all accidental or malicious damage.

**Configuration**

It is IMPLEMENTATION DEFINED whether EDLSR is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

If ARMv8.3-DoPD is implemented, then ARMv8.0-SoftwareLock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If ARMv8.3-DoPD is not implemented, then this register is in the Debug power domain.

Software uses EDLAR to set or clear the lock, and EDLSR to check the current status of the lock.

**Attributes**

EDLSR is a 32-bit register.

**Field descriptions**

The EDLSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>nTT, bit [2]</td>
</tr>
<tr>
<td>29</td>
<td>Not thirty-two bit access required. RAZ.</td>
</tr>
<tr>
<td>28</td>
<td>SLK, bit [1]</td>
</tr>
<tr>
<td>27</td>
<td>When the Software Lock is implemented.:</td>
</tr>
<tr>
<td>26</td>
<td>Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when the Software Lock is not implemented, this field is RES0.</td>
</tr>
<tr>
<td>25</td>
<td>For memory-mapped accesses when the Software Lock is implemented, possible values of this field are:</td>
</tr>
</tbody>
</table>

Page 3419
SLK, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ. For memory-mapped accesses, the value of this field is IMPLEMENTATION DEFINED. Permitted values are:

<table>
<thead>
<tr>
<th>SLI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Software Lock not implemented or not memory-mapped access.</td>
</tr>
<tr>
<td>0b1</td>
<td>Software Lock implemented and memory-mapped access.</td>
</tr>
</tbody>
</table>

Accessing the EDLSR

EDLSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFB4</td>
<td>EDLSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDPCSR, External Debug Program Counter Sample Register

The EDPCSR characteristics are:

**Purpose**

Holds a sampled instruction address value.

**Configuration**

EDPCSR is in the Core power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.0-PCSample is implemented and ARMv8.2-PCSample is not implemented. Otherwise, direct accesses to EDPCSR are **RES0**.

Implemented only if the **OPTIONAL** PC Sample-based Profiling Extension is implemented in the external debug registers space.

**Note**

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

**Attributes**

EDPCSR is a pair of 32-bit registers.

If ARMv8.1-VHE is implemented, the format of this register differs depending on the value of EDSCR.SC2.

**Field descriptions**

The EDPCSR bit assignments are:

**When ARMv8.1-VHE is not implemented or EDSCR.SC2 == 0:**

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**PC Sample high word, EDPCSRhi**

**PC Sample low word**

**Bits [63:32]**

PC Sample high word, EDPCSRhi. If EDVIDSR.HV == 0 then this field is RAZ, otherwise bits [63:32] of the sampled instruction address value. The translation regime that EDPCSR samples can be determined from EDVIDSR. {NS,E2,E3}.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [31:0]**

PC Sample low word. EDPCSRlo, bits[31:0] of the sampled instruction address value.

EDPCSRlo reads as 0xFFFFFFFF when any of the following are true:
The PE is in Debug state.
PC Sample-based profiling is prohibited.

If an instruction has retired since the PE left Reset state, then the first read of EDPCSR[31:0] is permitted but not required to return 0xFFFFFFFF.

EDPCSRlo reads as an UNKNOWN value when any of the following are true:

- The PE is in Reset state.
- No instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No instruction has retired since the last read of EDPCSR[31:0].

For the cases where a read of EDPCSR[31:0] returns 0xFFFFFFFF or an UNKNOWN value, the read has the side-effect of setting EDPCSRhi, EDCIDSR, and EDVIDSR to UNKNOWN values.

Otherwise, a read of EDPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to EDPCSRhi, EDCIDSR, and EDVIDSR. The translation regime that EDPCSR samples can be determined from EDVIDSR. {NS,E2,E3}.

For a read of EDPCSR[31:0] from the memory-mapped interface, if EDLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and EDPCSRhi, EDCIDSR, and EDVIDSR are unchanged.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ARMv8.1-VHE is implemented and EDSCR.SC2 == 1:

<table>
<thead>
<tr>
<th>NS</th>
<th>EL</th>
<th>RES0</th>
<th>PC Sample high word, EDPCSRhi</th>
<th>PC Sample low word</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
</tr>
</tbody>
</table>

**NS, bit [63]**

Non-secure state sample. Indicates the Security state that is associated with the most recent EDPCSR sample or, when it is read as a single atomic 64-bit read, the current EDPCSR sample. The translation regime that EDPCSR samples can be determined from EDPCSR. {NS,EL}.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample is from Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample is from Non-secure state.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**EL, bits [62:61]**

Exception level status sample. Indicates the Exception level that is associated with the most recent EDPCSR sample or, when it is read as a single atomic 64-bit read, the current EDPCSR sample. The translation regime that EDPCSR samples can be determined from EDPCSR. {NS,EL}.

<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Sample is from EL0.</td>
</tr>
<tr>
<td>0b01</td>
<td>Sample is from EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>Sample is from EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>Sample is from EL3.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [60:56]**

Reserved, RES0.
Bits [55:32]

PC Sample high word, EDPCSRhi. Bits [55:32] of the sampled instruction address value. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

PC Sample low word. EDPCSRlo, bits[31:0] of the sampled instruction address value.

EDPCSRlo reads as 0xFFFFFFFF when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If an instruction has retired since the PE left Reset state, then the first read of EDPCSR[31:0] is permitted but not required to return 0xFFFFFFFF.

EDPCSRlo reads as an UNKNOWN value when any of the following are true:

- The PE is in Reset state.
- No instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No instruction has retired since the last read of EDPCSR[31:0].

For the cases where a read of EDPCSR[31:0] returns 0xFFFFFFFF or an UNKNOWN value, the read has the side-effect of setting EDPCSRhi, EDCIDSR, and EDVIDSR to UNKNOWN values.

Otherwise, a read of EDPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to EDPCSRhi, EDCIDSR, and EDVIDSR. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

For a read of EDPCSR[31:0] from the memory-mapped interface, if EDLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and EDPCSRhi, EDCIDSR, and EDVIDSR are unchanged.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the EDPCSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

EDPCSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x0A0</td>
<td>EDPCSR</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x0AC</td>
<td>EDPCSR</td>
<td>63:32</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDPFR, External Debug Processor Feature Register

The EDPFR characteristics are:

**Purpose**

Provides information about implemented PE features.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

It is IMPLEMENTATION DEFINED whether EDPFR is implemented in the Core power domain or in the Debug power domain.

**Attributes**

EDPFR is a 64-bit register.

**Field descriptions**

The EDPFR bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| UNKNOWN| UNKNOWN| RES0   | UNKNOWN| AMU    | UNKNOWN| SEL2   | SVE    |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| UNKNOWN| GIC    | AdvSIMD| FP     | EL3    | EL2    | EL1    | EL0    |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |

**Bits [63:60]**

From Armv8.5:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

**Bits [59:56]**

From Armv8.5:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

**Bits [55:52]**

Reserved, RES0.
Bits [51:48]
From Armv8.4:
Reserved, UNKNOWN.

Otherwise:
Reserved, RES0.

AMU, bits [47:44]
From Armv8.4:
Activity Monitors Extension. Defined values are:

<table>
<thead>
<tr>
<th>AMU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Activity Monitors Extension is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Activity Monitors Extension version 1 is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.4-AMUv1 implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, Armv8.2, and Armv8.3, the only permitted value is 0b0000.

From Armv8.4, the permitted values are 0b0000 and 0b0001.

Otherwise:
Reserved, RES0.

Bits [43:40]
From Armv8.2:
Reserved, UNKNOWN.

Otherwise:
Reserved, RES0.

SEL2, bits [39:36]
From Armv8.4:
Secure EL2. Defined values are:

<table>
<thead>
<tr>
<th>SEL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Secure EL2 is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Secure EL2 is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:
Reserved, RES0.
SVE, bits [35:32]

From Armv8.2:

Scalable Vector Extension. Defined values are:

<table>
<thead>
<tr>
<th>SVE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>SVE is not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>SVE is implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

Bits [31:28]

From Armv8.2:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

GIC, bits [27:24]

System register GIC interface support. Defined values are:

<table>
<thead>
<tr>
<th>GIC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>GIC CPU interface system registers not implemented.</td>
</tr>
<tr>
<td>0b0001</td>
<td>System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported.</td>
</tr>
<tr>
<td>0b0011</td>
<td>System register interface to version 4.1 of the GIC CPU interface is supported.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.GIC.

AdvSIMD, bits [23:20]

Advanced SIMD. Defined values are:

<table>
<thead>
<tr>
<th>AdvSIMD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Advanced SIMD is implemented, including support for the following SISD and SIMD operations:</td>
</tr>
<tr>
<td></td>
<td>• Integer byte, halfword, word and doubleword element operations.</td>
</tr>
<tr>
<td></td>
<td>• Single-precision and double-precision floating-point arithmetic.</td>
</tr>
<tr>
<td></td>
<td>• Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As for 0b0000, and also includes support for half-precision floating-point arithmetic.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Advanced SIMD is not implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field must have the same value as the FP field.
The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support, that does not include the ARMv8.2-FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support, that includes the ARMv8.2-FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.AdvSIMD.

**FP, bits [19:16]**

Floating-point. Defined values are:

<table>
<thead>
<tr>
<th>FP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Floating-point is implemented, and includes support for:</td>
</tr>
<tr>
<td></td>
<td>• Single-precision and double-precision floating-point types.</td>
</tr>
<tr>
<td></td>
<td>• Conversions between single-precision and half-precision data types, and double-precision and half-precision data types.</td>
</tr>
<tr>
<td>0b0001</td>
<td>As for 0b0000, and also includes support for half-precision floating-point arithmetic.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Floating-point is not implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- 0b0000 in an implementation with floating-point support, that does not include the ARMv8.2-FP16 extension.
- 0b0001 in an implementation with floating-point support, that includes the ARMv8.2-FP16 extension.
- 0b1111 in an implementation without floating-point support.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.FP.

**EL3, bits [15:12]**

AArch64 EL3 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL3 is not implemented or cannot be executed in AArch64 state.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL3 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL3 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

When the value of EDAA32PFR.EL3 is non-zero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of ID_AA64PFR0_EL1.EL3.

**EL2, bits [11:8]**

AArch64 EL2 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL2 is not implemented or cannot be executed in AArch64 state.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL2 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL2 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

When the value of EDAA32PFR.EL2 is non-zero, this field must be 0b0000.

All other values are reserved.
In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of `ID_AA64PFR0_EL1_EL.EL2`.

**EL1, bits [7:4]**

AArch64 EL1 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL1 cannot be executed in AArch64 state. EL1 can be executed in AArch32 state only.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL1 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL1 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of `ID_AA64PFR0_EL1_EL.EL1`.

**EL0, bits [3:0]**

AArch64 EL0 Exception level handling. Defined values are:

<table>
<thead>
<tr>
<th>EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>EL0 cannot be executed in AArch64 state. EL0 can be executed in AArch32 state only.</td>
</tr>
<tr>
<td>0b0001</td>
<td>EL0 can be executed in AArch64 state only.</td>
</tr>
<tr>
<td>0b0010</td>
<td>EL0 can be executed in either AArch64 or AArch32 state.</td>
</tr>
</tbody>
</table>

All other values are reserved.

In an Armv8-A implementation that supports AArch64 state in at least one Exception level, this field returns the value of `ID_AA64PFR0_EL1_EL.EL0`.

**Accessing the EDPFR**

**EDPFR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xD20</td>
<td>EDPFR</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()` and `!DoubleLockStatus()` accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xD24</td>
<td>EDPFR</td>
<td>63:32</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()` and `!DoubleLockStatus()` accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.
EDPIDR0, External Debug Peripheral Identification Register 0

The EDPIDR0 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDPIDR0 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDPIDR0 is a 32-bit register.

**Field descriptions**

The EDPIDR0 bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9  | Bit 8  | Bit 7  | Bit 6  | Bit 5  | Bit 4  | Bit 3  | Bit 2  | Bit 1  | Bit 0  |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        | RES0   |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |
| **RES0** |        |        |        |        |        |        |        |        |        | **PART_0** |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |

**Bits [31:8]**

Reserved, RES0.

**PART_0, bits [7:0]**

Part number, least significant byte.

**Accessing the EDPIDR0**

EDPIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFE0</td>
<td>EDPIDR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
EDPIDR1, External Debug Peripheral Identification Register 1

The EDPIDR1 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is IMPLEMENTATION DEFINED whether EDPIDR1 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDPIDR1 is a 32-bit register.

**Field descriptions**

The EDPIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>27-24</td>
<td>DES_0</td>
</tr>
<tr>
<td>20-16</td>
<td>PART_1</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**DES_0, bits [7:4]**

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is 0b1011.

**PART_1, bits [3:0]**

Part number, most significant nibble.

**Accessing the EDPIDR1**

EDPIDR1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFE4</td>
<td>EDPIDR1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
• Otherwise accesses to this register generate an error response.
EDPIDR2, External Debug Peripheral Identification Register 2

The EDPIDR2 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is IMPLEMENTATION DEFINED whether EDPIDR2 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDPIDR2 is a 32-bit register.

**Field descriptions**

The EDPIDR2 bit assignments are:

```
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  |  9  |  8  |  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| REVISION | JEDEC | DES_1 |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

JEDEC, bit [3]

RAO. Indicates a JEP106 identity code is used.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is 0b011.
Accessing the EDPIDR2

EDPIDR2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFE8</td>
<td>EDPIDR2</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDPIDR3, External Debug Peripheral Identification Register 3

The EDPIDR3 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDPIDR3 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDPIDR3 is a 32-bit register.

**Field descriptions**

The EDPIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>REVAND</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CMOD</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Reserved, RES0.

**REVAND, bits [7:4]**

Part minor revision. Parts using EDPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

**CMOD, bits [3:0]**

Customer modified. Indicates someone other than the Designer has modified the component.

**Accessing the EDPIDR3**

EDPIDR3 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFEC</td>
<td>EDPIDR3</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
EDPIDR4, External Debug Peripheral Identification Register 4

The EDPIDR4 characteristics are:

**Purpose**

Provides information to identify an external debug component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether EDPIDR4 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

EDPIDR4 is a 32-bit register.

**Field descriptions**

The EDPIDR4 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

**Bits [31:8]**

Reserved, RES0.

**SIZE, bits [7:4]**

Size of the component. RAZ. Log2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

**DES_2, bits [3:0]**

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is 0b0100.

**Accessing the EDPIDR4**

EDPIDR4 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xFD0</td>
<td>EDPIDR4</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDPRCR, External Debug Power/Reset Control Register

The EDPRCR characteristics are:

**Purpose**

Controls the PE functionality related to powerup, reset, and powerdown.

**Configuration**

EDPRCR contains fields that are in the Core power domain and fields that are in the Debug power domain.

For RW fields see the field description for a description of the behavior of the field on a reset that applies to its power domain. However:

- Fields that are in the Core power domain are not affected by a warm reset and are not affected by an External debug reset.
- Fields that are in the Debug power domain reset to their defined reset values on an External debug reset, and are not affected by a Warm reset and are not affected by a Cold reset.

If ARMv8.3-DoPD is implemented then all fields in this register are in the Core power domain.

CORENPDRQ is the only field that is mapped between the EDPRCR and DBGPRCR and DBGPRCR_EL1.

**Attributes**

EDPRCR is a 32-bit register.

**Field descriptions**

The EDPRCR bit assignments are:

**When ARMv8.3-DoPD is implemented:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CWRR</td>
<td>CORENPDRQ</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:2]**

Reserved, RES0.

**CWRR, bit [1]**

Warm reset request. Write-only bit that reads as zero.

The extent of the reset is IMPLEMENTATION DEFINED, but must be one of:

- The request is ignored.
- Only this PE is Warm reset.
- This PE and other components of the system, possibly including other PEs, are Warm reset.

Arm deprecates use of this bit, and recommends that implementations ignore the request.

<table>
<thead>
<tr>
<th>CWRR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Request Warm reset.</td>
</tr>
</tbody>
</table>
This field is in the Core power domain.

The PE ignores writes to this bit if any of the following are true:

- `ExternalInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Non-secure state.`
- `ExternalSecureInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Secure state.`
- `ExternalSecureInvasiveDebugEnabled() == FALSE and EL3 is implemented.`

In an implementation that includes the recommended external debug interface, this bit drives the DBGRSTREQ signal.

On a Warm reset, this field resets to 0.

Accessing this field has the following behavior:

- When `!IsCorePowered(), or DoubleLockStatus(), or OSLockStatus() or SoftwareLockStatus()`, access to this field is `WI`.
- Otherwise, access to this field is `WO`.

**CORENPDRQ, bit [0]**

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the **IMPLEMENTATION DEFINED** nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>CORENPDRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the system responds to a powerdown request, it powers down Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the system responds to a powerdown request, it does not power down the Core power domain, but instead emulates a powerdown of that domain.</td>
</tr>
</tbody>
</table>

When this bit reads as `UNKNOWN`, the PE ignores writes to this bit.

This field is in the Core power domain, and permitted accesses to this field map to the `DBGPRC_EL1.CORENPDRQ` and `DBGPRC_EL1.CORENPDRQ` fields.

In an implementation that includes the recommended external debug interface, this bit drives the `DBGNOPWRDWN` signal.

It is **IMPLEMENTATION DEFINED** whether this bit is reset the Cold reset value on exit from an **IMPLEMENTATION DEFINED** software-visible retention state. For more information about retention states see Core power domain power states.

**Note**

Writes to this bit are not prohibited by the **IMPLEMENTATION DEFINED** authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The following resets apply:

- On a Cold reset, this field is set to 1 if the powerup request is implemented and the powerup request has been asserted, and is set to 0 otherwise.
- On a Warm reset, the value of this field is unchanged.

Accessing this field has the following behavior:

- When `!IsCorePowered(), or DoubleLockStatus() or OSLockStatus()`, access to this field is `UNKNOWN`.
- When `SoftwareLockStatus()`, access to this field is `RO`.
- Otherwise, access to this field is `RW`.
Otherwise:

<table>
<thead>
<tr>
<th>Bits [31:4]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
</table>

**COREPURQ, bit [3]**

Core powerup request. Allows a debugger to request that the power controller power up the core, enabling access to the debug register in the Core power domain, and that the power controller emulates powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the *IMPLEMENTATION DEFINED* nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>COREPURQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not request power up of the Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>Request power up of the Core power domain, and emulation of powerdown.</td>
</tr>
</tbody>
</table>

In an implementation that includes the recommended external debug interface, this bit drives the DBGPWRUPREQ signal.

This field is in the Debug power domain and can be read and written when the Core power domain is powered off.

**Note**

 Writes to this bit are not prohibited by the *IMPLEMENTATION DEFINED* authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a External debug reset, this field resets to 0.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is RO.
- Otherwise, access to this field is RW.

**Bit [2]**

Reserved, RES0.

**CWRR, bit [1]**

Warm reset request. Write-only bit that reads as zero.

The extent of the reset is *IMPLEMENTATION DEFINED*, but must be one of:

- The request is ignored.
- Only this PE is Warm reset.
- This PE and other components of the system, possibly including other PEs, are Warm reset.

Arm deprecates use of this bit, and recommends that implementations ignore the request.

<table>
<thead>
<tr>
<th>CWRR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Request Warm reset.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain

The PE ignores writes to this bit if any of the following are true:
**External Invasive Debug Enabled() == FALSE, EL3 is not implemented, and the implemented Security state is Non-secure state.**

**External Secure Invasive Debug Enabled() == FALSE, EL3 is not implemented, and the implemented Security state is Secure state.**

**External Secure Invasive Debug Enabled() == FALSE and EL3 is implemented.**

In an implementation that includes the recommended external debug interface, this bit drives the DBGRSTREQ signal.

On a Warm reset, this field resets to 0.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus(), or OSLockStatus() or SoftwareLockStatus(), access to this field is **WI**.
- Otherwise, access to this field is **WO**.

**CORENPDRO, bit [0]**

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

<table>
<thead>
<tr>
<th>CORENPDRO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the system responds to a powerdown request, it powers down Core power domain.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain.</td>
</tr>
</tbody>
</table>

When this bit reads as **UNKNOWN**, the PE ignores writes to this bit.

This field is in the Core power domain, and permitted accesses to this field map to the **DBGPRCR.CORENPDRO** and **DBGPRCR_EL1.CORENPDRO** fields.

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is **IMPLEMENTATION DEFINED** whether this bit is reset to the value of **EDPRCR.COREPURQ** on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see Core power domain power states.

**Note**

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The following resets apply:

- On a Cold reset, this field resets to the value in **EDPRCR.COREPURQ**.
- On a Warm reset, the value of this field is unchanged.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or OSLockStatus() or SoftwareLockStatus(), access to this field is **UNKNOWN**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

**Accessing the EDPRCR**

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.
**EDPRCR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x310</td>
<td>EDPRCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When (ARMv8.3-DoPD is not implemented or IsCorePowered()) and SoftwareLockStatus() accesses to this register are **RO**.
- When (ARMv8.3-DoPD is not implemented or IsCorePowered()) and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
EDPRSR, External Debug Processor Status Register

The EDPRSR characteristics are:

Purpose

Holds information about the reset and powerdown state of the PE.

Configuration

EDPRSR contains fields that are in the Core power domain and fields that are in the Debug power domain.

Some of the fields in the Core power domain are in the Cold reset domain and others are in the Warm reset domain. See the field descriptions for more information. However:

- Fields that are in the Cold reset domain are not affected by a warm reset and are not affected by an External debug reset.
- Fields in the Warm reset domain are also reset by a Cold reset but are not affected by an External debug reset.
- Fields in the Debug power domain are not affected by a Warm reset and are not affected by a Cold reset.

If ARMv8.3-DoPD is implemented then all fields in this register are in the Core power domain.

Attributes

EDPRSR is a 32-bit register.

Field descriptions

The EDPRSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | STAD | ETAD | SDR | PMADE | SDADE | DAD | DLK | OSLK | HALTED | SR | SPD | PU |

Bits [31:14]

Reserved, RES0.

STAD, bit [13]

When TRBE is implemented:

Sticky ETAD error. Set to 1 when a Non-secure external debug interface access to an external PE Trace Unit register returns an error because AllowExternalTraceAccess() == FALSE for the access.

<table>
<thead>
<tr>
<th>STAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Non-secure external debug interface accesses to the PE Trace Unit registers have failed because AllowExternalTraceAccess() == FALSE for the access since EDPRSR was last read.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one Non-secure external debug interface access to the PE Trace Unit registers has failed and returned an error because AllowExternalTraceAccess() == FALSE for the access since EDPRSR was last read.</td>
</tr>
</tbody>
</table>

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE then this bit clears to 0.
EDPRS R, External Debug Processor Status Register

- If ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE then it is **CONSTRAINED UNPREDICTABLE** whether this bit clears to 0 or is unchanged.

This bit is in the Core power domain.

On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRS R.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

**Otherwise:**

Reserved, RES0.

**ETAD, bit [12]**

**When TRBE is implemented:**

External Trace Access Disable status.

<table>
<thead>
<tr>
<th>ETAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External Non-secure PE Trace Unit accesses enabled. AllowExternalTraceAccess() == TRUE for a Non-secure access.</td>
</tr>
<tr>
<td>0b1</td>
<td>External Non-secure PE Trace Unit accesses disabled. AllowExternalTraceAccess() == FALSE for a Non-secure access.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRS R.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

**Otherwise:**

Reserved, RES0.

**SDR, bit [11]**

Sticky Debug Restart. Set to 1 when the PE exits Debug state.

Permitted values are:

<table>
<thead>
<tr>
<th>SDR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PE has not restarted since EDPRS R was last read.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PE has restarted since EDPRS R was last read.</td>
</tr>
</tbody>
</table>

**Note**

If a reset occurs when the PE is in Debug state, the PE exits Debug state. SDR is **UNKNOWN** on Warm reset, meaning a debugger must also use the SR bit to determine whether the PE has left Debug state.

If The Core power domain is powered up, then following a read of EDPRS R:

- If ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is **CONSTRAINED UNPREDICTABLE** whether this bit clears to 0 or is unchanged.

This field is in the Core power domain and the Warm reset domain.

This field resets to an architecturally **UNKNOWN** value.
Accessing this field has the following behavior:

- When `!IsCorePowered()`, or `DoubleLockStatus()` or `EDPRSR.R == 1`, access to this field is **UNKNOWN**.
- When `SoftwareLockStatus()`, access to this field is **RO**.
- Otherwise, access to this field is **RC**.

**SPMAD, bit [10]**

*When ARMv8.4-Debug is implemented:*

Sticky EPMAD error. Set to 1 if an external debug interface access to a Performance Monitors register returns an error because `AllowExternalPMUAccess() == FALSE`.

Permitted values are:

<table>
<thead>
<tr>
<th>SPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Non-secure external debug interface accesses to the external Performance Monitors registers have failed because <code>AllowExternalPMUAccess() == FALSE</code> for the access since <code>EDPRSR</code> was last read.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one Non-secure external debug interface access to the external Performance Monitors register has failed and returned an error because <code>AllowExternalPMUAccess() == FALSE</code> for the access since <code>EDPRSR</code> was last read.</td>
</tr>
</tbody>
</table>

If the Core power domain is powered up, then, following a read of `EDPRSR`:

- If ARMv8.0-DoubleLock is not implemented or `DoubleLockStatus() == FALSE`, this bit clears to 0.
- If ARMv8.0-DoubleLock is implemented, and `DoubleLockStatus() == TRUE`, it is **CONSTRAINED UNPREDICTABLE** whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- When `!IsCorePowered()`, or `DoubleLockStatus()` or `EDPRSR.R == 1`, access to this field is **UNKNOWN**.
- When `SoftwareLockStatus()`, access to this field is **RO**.
- Otherwise, access to this field is **RC**.

**Otherwise:**

Sticky EPMAD error.

<table>
<thead>
<tr>
<th>SPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No external debug interface accesses to the Performance Monitors registers have failed because <code>AllowExternalPMUAccess() == FALSE</code> since <code>EDPRSR</code> was last read.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one external debug interface access to the Performance Monitors registers has failed and returned an error because <code>AllowExternalPMUAccess() == FALSE</code> since <code>EDPRSR</code> was last read.</td>
</tr>
</tbody>
</table>

If the Core power domain is powered up, then, following a read of `EDPRSR`:

- If ARMv8.0-DoubleLock is not implemented or `DoubleLockStatus() == FALSE`, this bit clears to 0.
- If ARMv8.0-DoubleLock is implemented, and `DoubleLockStatus() == TRUE`, it is **CONSTRAINED UNPREDICTABLE** whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:
• When !IsCorePowered(), or OSLockStatus(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
• When SoftwareLockStatus(), access to this field is **RO**.
• Otherwise, access to this field is **RC**.

**EPMAD, bit [9]**

*When ARMv8.4-Debug is implemented:*

External Performance Monitors Access Disable status.

<table>
<thead>
<tr>
<th>EPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External Non-secure Performance Monitors access enabled. AllowExternalPMUAccess() == TRUE for a Non-secure access.</td>
</tr>
<tr>
<td>0b1</td>
<td>External Non-secure Performance Monitors access disabled. AllowExternalPMUAccess() == FALSE for a Non-secure access.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain.

Accessing this field has the following behavior:

• When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
• Otherwise, access to this field is **RO**.

**Otherwise:**

External Performance Monitors access disable status.

<table>
<thead>
<tr>
<th>EPMAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External Performance Monitors access enabled. AllowExternalPMUAccess() == TRUE.</td>
</tr>
<tr>
<td>0b1</td>
<td>External Performance Monitors access disabled. AllowExternalPMUAccess() == FALSE.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain.

Accessing this field has the following behavior:

• When !IsCorePowered(), or OSLockStatus(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
• Otherwise, access to this field is **RO**.

**SDAD, bit [8]**

*When ARMv8.4-Debug is implemented:*

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because AllowExternalDebugAccess() == FALSE.

<table>
<thead>
<tr>
<th>SDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No Non-secure external debug interface accesses to the debug registers have failed because AllowExternalDebugAccess() == FALSE for the access since EDPRSR was last read.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one Non-secure external debug interface access to the debug registers has failed and returned an error because AllowExternalDebugAccess() == FALSE for the access since EDPRSR was last read.</td>
</tr>
</tbody>
</table>

If the Core power domain is powered up, then, following a read of EDPRSR:

• If ARMv8.0[DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.  
• If ARMv8.0[DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.  

This field is in the Core power domain.
On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

Otherwise:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because AllowExternalDebugAccess() == FALSE.

<table>
<thead>
<tr>
<th>SDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No external debug interface accesses to the debug registers have failed because AllowExternalDebugAccess() == FALSE since EDPRSR was last read.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one external debug interface access to the debug registers has failed and returned an error because AllowExternalDebugAccess() == FALSE since EDPRSR was last read.</td>
</tr>
</tbody>
</table>

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is **CONSTRAINED UNPREDICTABLE** whether this bit clears to 0 or is unchanged.

This bit is **UNKNOWN** on reads if OSLockStatus() == TRUE and external debug writes to OSLAR_EL1 do not return an error when AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

**EDAD, bit [7]**

When ARMv8.4-Debug is implemented:

External Debug Access Disable status.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 enabled. AllowExternalDebugAccess() == TRUE for a Non-secure access.</td>
</tr>
<tr>
<td>0b1</td>
<td>External Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 disabled. AllowExternalDebugAccess() == FALSE for a Non-secure access.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered(), or DoubleLockStatus() or EDPRSR.R == 1, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

When ARMv8.2-Debug is implemented:

External Debug Access Disable status.
### EDPRSR, External Debug Processor Status Register

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External access to breakpoint registers, watchpoint registers, and <code>OSLR_EL1</code> enabled. <code>AllowExternalDebugAccess()</code> == TRUE.</td>
</tr>
<tr>
<td>0b1</td>
<td>External access to breakpoint registers, watchpoint registers, and <code>OSLR_EL1</code> disabled. <code>AllowExternalDebugAccess()</code> == FALSE.</td>
</tr>
</tbody>
</table>

This bit is not valid and reads **UNKNOWN** if `OSLockStatus()` == TRUE and external debug writes to `OSLR_EL1` do not return an error when `AllowExternalDebugAccess()` == FALSE.

This field is in the Core power domain. Accessing this field has the following behavior:

- When `!IsCorePowered()`, or `DoubleLockStatus()` or `EDPRSR.R == 1`, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

**Otherwise:**

External Debug Access Disable status.

<table>
<thead>
<tr>
<th>EDAD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External access to breakpoint registers, watchpoint registers, and <code>OSLR_EL1</code> enabled. <code>AllowExternalDebugAccess()</code> == TRUE.</td>
</tr>
<tr>
<td>0b1</td>
<td>External access to breakpoint registers, watchpoint registers disabled. It is <strong>IMPLEMENTATION DEFINED</strong> whether accesses to <code>OSLR_EL1</code> are enabled or disabled. <code>AllowExternalDebugAccess()</code> == FALSE.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain. Accessing this field has the following behavior:

- When `!IsCorePowered()`, or `DoubleLockStatus()` or `EDPRSR.R == 1`, access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

**DLK, bit [6]**

**When ARMv8.4-Debug is implemented:**

This field is **RES0**.

**When ARMv8.2-Debug is implemented and ARMv8.0-DoubleLock is implemented:**

Double Lock.

From Armv8.2, this field is deprecated.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When `IsCorePowered()` and `!DoubleLockStatus()`, access to this field is **RAZ**.
- Otherwise, access to this field is **UNKNOWN**.

**When ARMv8.0-DoubleLock is implemented:**

Double Lock.

This field returns the result of the pseudocode function `DoubleLockStatus()`.

If the Core power domain is powered up and `DoubleLockStatus()` == TRUE, it is **IMPLEMENTATION DEFINED** whether:
EDPRSR, External Debug Processor Status Register

- EDPRSR.PU reads as 1, EDPRSR.DLK reads as 1, and EDPRSR.SPD is \textit{UNKNOWN}.
- EDPRSR.PU reads as 0, EDPRSR.DLK is \textit{UNKNOWN}, and EDPRSR.SPD reads as 0.

This field is in the Core power domain.

<table>
<thead>
<tr>
<th>DLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>DoubleLockStatus() returns FALSE.</td>
</tr>
<tr>
<td>0b1</td>
<td>DoubleLockStatus() returns TRUE and the Core power domain is powered up.</td>
</tr>
</tbody>
</table>

Accessing this field has the following behavior:

- When \texttt{!IsCorePowered()}, access to this field is \textit{UNKNOWN}.
- Otherwise, access to this field is \textit{RO}.

\textbf{Otherwise:}

Reserved, RES0.

\textbf{OSLK, bit [5]}

OS Lock status bit.

A read of this bit returns the value of \texttt{OSLSR_EL1.OSLK}.

This field is in the Core power domain.

Accessing this field has the following behavior:

- When \texttt{!IsCorePowered()}, DoubleLockStatus() and EDPRSR.R == 1, access to this field is \textit{UNKNOWN}.
- Otherwise, access to this field is \textit{RO}.

\textbf{HALTED, bit [4]}

Halted status bit.

<table>
<thead>
<tr>
<th>HALTED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PE is in Non-debug state.</td>
</tr>
<tr>
<td>0b1</td>
<td>PE is in Debug state.</td>
</tr>
</tbody>
</table>

This field is in the Core power domain.

Accessing this field has the following behavior:

- When \texttt{!IsCorePowered()}, access to this field is \textit{UNKNOWN}.
- Otherwise, access to this field is \textit{RO}.

\textbf{SR, bit [3]}

Sticky core Reset status bit.

Permitted values are:

<table>
<thead>
<tr>
<th>SR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The non-debug logic of the PE is not in reset state and has not been reset since the last time EDPRSR was read.</td>
</tr>
<tr>
<td>0b1</td>
<td>The non-debug logic of the PE is in reset state or has been reset since the last time EDPRSR was read.</td>
</tr>
</tbody>
</table>

If EDPRSR.PU reads as 1 and EDPRSR.R reads as 0, which means that the Core power domain is in a powerup state and that the non-debug logic of the PE is not in reset state, then following a read of EDPRSR:

- If ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is \textit{UNPREDICTABLE} whether this bit clears to 0 or is unchanged.
This field is in the Core power domain and the Warm reset domain.

This field resets to 1.

Accessing this field has the following behavior:

- When !IsCorePowered() or DoubleLockStatus(), access to this field is **UNKNOWN**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC**.

**R, bit [2]**

PE Reset status bit.

Permitted values are:

<table>
<thead>
<tr>
<th>R</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The non-debug logic of the PE is not in reset state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The non-debug logic of the PE is in reset state.</td>
</tr>
</tbody>
</table>

If ARMv8.0-DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a **CONSTRAINED UNPREDICTABLE** value. For more information see 'EDPRSR.{DLK, R} and reset state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

This field is in the Core power domain.

Accessing this field has the following behavior:

- When !IsCorePowered() or DoubleLockStatus(), access to this field is **UNKNOWN**.
- Otherwise, access to this field is **RO**.

**SPD, bit [1]**

Sticky core Powerdown status bit.

If ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, then:

- If ARMv8.2-Debug is implemented, this bit reads as 0.
- If ARMv8.2-Debug is not implemented, this bit might read as 0 or 1.

For more information, see EDPRSR.{DLK, SPD, PU} and the Core power domain.

<table>
<thead>
<tr>
<th>SPD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If EDPRSR.PU is 0, it is not known whether the state of the debug registers in the Core power domain is lost. If EDPRSR.PU is 1, the state of the debug registers in the Core power domain has not been lost.</td>
</tr>
<tr>
<td>0b1</td>
<td>The state of the debug registers in the Core power domain has been lost.</td>
</tr>
</tbody>
</table>

If the Core power domain is powered up, then, following a read of EDPRSR:

- If ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, it is **CONSTRAINED UNPREDICTABLE** whether this bit clears to 0 or is unchanged.

When the Core power domain is in either retention or powerdown state, the value of EDPRSR.SPD is **IMPLEMENTATION DEFINED**. For more information, see 'EDPRSR.SPD when the Core domain is in either retention or powerdown state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support).

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} bits record accessibility and lost of state in Core power domain' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support).

This field is in the Core power domain and the Cold reset domain.
On a Cold reset, this field resets to 1.

Accessing this field has the following behavior:

- When !IsCorePowered(), access to this field is RAZ.
- When IsCorePowered() and DoubleLockStatus(), access to this field is UNKNOWN.
- Otherwise, access to this field is RO.

**PU, bit [0]**

*When ARMv8.3-DoPD is implemented:*

Core powerup status bit.

Access to this field is RAO.

*When ARMv8.2-Debug is implemented:*

Core Powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

<table>
<thead>
<tr>
<th>PU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Either the Core power domain is in a low-power or powerdown state, or ARMv8.0-DoubleLock is implemented and DoubleLockStatus() == TRUE, meaning the debug registers in the Core power domain cannot be accessed.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Core power domain is in a powerup state, and either ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE, meaning the debug registers in the Core power domain can be accessed.</td>
</tr>
</tbody>
</table>

If ARMv8.0-DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONstrained UNpredicTaBle value. For more information see 'EDPRSR.{DLK, R} and reset state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} bits record accessibility and lost of state in Core power domain' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

Access to this field is RO.

**Otherwise:**

Core Powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

When the Core power domain is powered-up and DoubleLockStatus() == TRUE, then the value of EDPRSR.PU is IMPLEMENTATION DEFINED. See the description of the DLK bit for more information.

Otherwise, permitted values are:

<table>
<thead>
<tr>
<th>PU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Core power domain is in a low-power or powerdown state where the debug registers in the Core power domain cannot be accessed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Core power domain is in a powerup state where the debug registers in the Core power domain can be accessed.</td>
</tr>
</tbody>
</table>

If ARMv8.0-DoubleLock is implemented, the Core power domain is powered up, and DoubleLockStatus() == TRUE, it is IMPLEMENTATION DEFINED whether this bit reads as 0 or 1.

If ARMv8.0-DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONstrained UNpredicTaBle value. For more information see 'EDPRSR.{DLK, R} and reset state' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} bits record accessibility and lost of state in Core power domain' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)
record accessibility and lost of state in Core power domain’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H6 (Debug Reset and Powerdown Support)

Access to this field is RO.

**Accessing the EDPRSR**

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.

If the Core power domain is powered up (EDPRSR.PU == 1), then following a read of EDPRSR:

- If ARMv8.0-DoubleLock is not implemented or DoubleLockStatus() == FALSE, then:
  - EDPRSR.{SDR, SPMAD, SDAD, SPD} are cleared to 0.
  - EDPRSR.SR is cleared to 0 if the non-debug logic of the PE is not in reset state (EDPRSR.R == 0).
- Otherwise it is CONSTRAINED UNPREDICTABLE whether or not this clearing occurs.

If the Core power domain is powered down (EDPRSR.PU == 0), then:

- EDPRSR.{SDR, SPMAD, SDAD, SR} are all UNKNOWN, and are either reset or restored on being powered up.
- EDPRSR.SPD is not cleared following a read of EDPRSR. See the SPD bit description for more information.

The clearing of bits is an indirect write to EDPRSR.

**EDPRSR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x314</td>
<td>EDPRSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDRCR, External Debug Reserve Control Register

The EDRCR characteristics are:

Purpose

This register is used to allow imprecise entry to Debug state and clear sticky bits in EDSCR.

Configuration

EDRCR is in the Core power domain.

Attributes

EDRCR is a 32-bit register.

Field descriptions

The EDRCR bit assignments are:

|    | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

Bits [31:5]

Reserved, RES0.

CBRRQ, bit [4]

Allow imprecise entry to Debug state. The actions on writing to this bit are:

<table>
<thead>
<tr>
<th>CBRRQ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Allow imprecise entry to Debug state, for example by canceling pending bus accesses.</td>
</tr>
</tbody>
</table>

Setting this bit to 1 allows a debugger to request imprecise entry to Debug state. An External Debug Request debug event must be pending before the debugger sets this bit to 1.

This feature is optional. If this feature is not implemented, writes to this bit are ignored.

CSPA, bit [3]

Clear Sticky Pipeline Advance. This bit is used to clear the EDSCR.PipeAdv bit to 0. The actions on writing to this bit are:

<table>
<thead>
<tr>
<th>CSPA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Clear the EDSCR.PipeAdv bit to 0.</td>
</tr>
</tbody>
</table>

CSE, bit [2]

Clear Sticky Error. Used to clear the EDSCR cumulative error bits to 0. The actions on writing to this bit are:
### CSE Meaning

<table>
<thead>
<tr>
<th>CSE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Clear the EDSCR {TXU, RXO, ERR} bits, and, if the PE is in Debug state, the EDSCR.IFO bit, to 0.</td>
</tr>
</tbody>
</table>

**Bits [1:0]**

Reserved, RES0.

### Accessing the EDRCR

EDRCR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x090</td>
<td>EDRCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are WI.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are WO.
- Otherwise accesses to this register generate an error response.
EDSCR, External Debug Status and Control Register

The EDSCR characteristics are:

**Purpose**

Main control register for the debug implementation.

**Configuration**

External register EDSCR bits [30:29] are architecturally mapped to AArch64 System register MDCCSR_EL0[30:29].

EDSCR is in the Core power domain. Some or all RW fields of this register have defined reset values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

**Attributes**

EDSCR is a 32-bit register.

**Field descriptions**

The EDSCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>TFO</td>
</tr>
<tr>
<td>30</td>
<td>RXfull</td>
</tr>
<tr>
<td>29</td>
<td>TXfull</td>
</tr>
<tr>
<td>28</td>
<td>TXO</td>
</tr>
<tr>
<td>27</td>
<td>TXPipeAdv</td>
</tr>
<tr>
<td>26</td>
<td>INTdis</td>
</tr>
<tr>
<td>25</td>
<td>ITE</td>
</tr>
<tr>
<td>24</td>
<td>ITO</td>
</tr>
<tr>
<td>23</td>
<td>MA</td>
</tr>
<tr>
<td>22</td>
<td>SC2</td>
</tr>
<tr>
<td>21</td>
<td>NS</td>
</tr>
<tr>
<td>20</td>
<td>HDE</td>
</tr>
<tr>
<td>19</td>
<td>SDD</td>
</tr>
<tr>
<td>18</td>
<td>RES0</td>
</tr>
<tr>
<td>17</td>
<td>RES0</td>
</tr>
<tr>
<td>16</td>
<td>RW</td>
</tr>
<tr>
<td>15</td>
<td>ELA</td>
</tr>
<tr>
<td>14</td>
<td>ERR</td>
</tr>
<tr>
<td>13</td>
<td>STATUS</td>
</tr>
</tbody>
</table>

**TFO, bit [31]**

Trace Filter Override. Overrides the Trace Filter controls allowing the external debugger to trace any visible Exception level.

<table>
<thead>
<tr>
<th>TFO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Trace Filter controls are not affected.</td>
</tr>
<tr>
<td>01</td>
<td>Trace Filter controls in TRFCR_EL1, TRFCR_EL2, TRFCR, and HTRFCR are ignored.</td>
</tr>
</tbody>
</table>

When OSLSR_EL1.OSLK == 1, this bit can be indirectly read and written through the MDSCR_EL1 and DBGDSRext System registers.

This bit is ignored by the PE when ExternalSecureNoninvasiveDebugEnabled() == FALSE and the Effective value of MDCR_EL3.STE == 1.

On a Cold reset, this field resets to 0.

**Other wise:**

Reserved, RES0.

**RXfull, bit [30]**

DTRRX full.

On a Cold reset, this field resets to 0.

Access to this field is RO.
**TXfull, bit [29]**

DRTTX full.

On a Cold reset, this field resets to 0.

Access to this field is **RO**.

**ITO, bit [28]**

ITR overrun.

If the PE is in Non-debug state, this bit is **UNKNOWN**. ITO is set to 0 on entry to Debug state.

Access to this field is **RO**.

**RXO, bit [27]**

DTRRX overrun.

On a Cold reset, this field resets to 0.

Access to this field is **RO**.

**TXU, bit [26]**

DRTTX underrun.

On a Cold reset, this field resets to 0.

Access to this field is **RO**.

**PipeAdv, bit [25]**

Pipeline advance. Set to 1 every time the PE pipeline retires one or more instructions. Cleared to 0 by a write to EDRCR.CSPA.

The architecture does not define precisely when this bit is set to 1. It requires only that this happen periodically in Non-debug state to indicate that software execution is progressing.

Access to this field is **RO**.

**ITE, bit [24]**

ITR empty.

If the PE is in Non-debug state, this bit is **UNKNOWN**. It is always valid in Debug state.

Access to this field is **RO**.

**INTdis, bits [23:22]**

**When ARMv8.4-Debug is implemented:**

Interrupt disable. Disables taking interrupts in Non-Debug state.

<table>
<thead>
<tr>
<th>INTdis</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Masking of interrupts is controlled by PSTATE and interrupt routing controls.</td>
</tr>
<tr>
<td>0b1</td>
<td>If ExternalSecureDebugEnabled() == TRUE, then all interrupts, including virtual and SError interrupts, are masked. If ExternalSecureDebugEnabled() == FALSE, then all interrupts targeting Non-secure state are masked.</td>
</tr>
</tbody>
</table>
When $\text{OSLSR\_EL1. OSLK} == 1$, this field can be indirectly read and written through the $\text{MDSCR\_EL1}$ and $\text{DBGDSCR\_Ext}$ System registers.

This field is ignored by the PE and treated as zero when ExternalDebugEnabled() == FALSE.

On a Cold reset, this field resets to 0.

**Otherwise:**

Interrupt disable.

When $\text{OSLSR\_EL1. OSLK} == 1$, this field can be indirectly read and written through the $\text{MDSCR\_EL1}$ and $\text{DBGDSCR\_Ext}$ System registers.

<table>
<thead>
<tr>
<th>INTdis</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Do not disable interrupts.</td>
</tr>
<tr>
<td>0b01</td>
<td>Disable interrupts taken to Non-secure EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>Disable interrupts taken only to Non-secure EL1 and Non-secure EL2. If ExternalSecureInvasiveDebugEnabled() == TRUE, also disable interrupts taken to Secure EL1.</td>
</tr>
<tr>
<td>0b11</td>
<td>Disable interrupts taken only to Non-secure EL1 and Non-secure EL2. If ExternalSecureInvasiveDebugEnabled() == TRUE, also disable all other interrupts.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**TDA, bit [21]**

Traps accesses to the following debug System registers:

- AArch64: $\text{DBGBCR<n>_EL1}, \text{DBGBVR<n>_EL1}, \text{DBGWCR<n>_EL1}, \text{DBGWVR<n>_EL1}$
- AArch32: $\text{DBGBCR<n>}, \text{DBGBVR<n>}, \text{DBGBXVR<n>}, \text{DBGWCR<n>}, \text{DBGWVR<n>}$.  

The possible values of this field are:

<table>
<thead>
<tr>
<th>TDA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Accesses to debug System registers do not generate a Software Access Debug event.</td>
</tr>
<tr>
<td>0b1</td>
<td>Accesses to debug System registers generate a Software Access Debug event, if $\text{OSLSR_EL1. OSLK}$ is 0 and if halting is allowed.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**MA, bit [20]**

Memory access mode. Controls the use of memory-access mode for accessing ITR and the DCC. This bit is ignored if in Non-debug state and set to zero on entry to Debug state.

Possible values of this field are:

<table>
<thead>
<tr>
<th>MA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal access mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory access mode.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**SC2, bit [19]**

When ARMv8.0-PCSample is implemented, ARMv8.1-VHE is implemented and ARMv8.2-PCSample is not implemented:

Sample $\text{CONTEXTIDR\_EL2}$. Controls whether the PC Sample-based Profiling Extension samples $\text{CONTEXTIDR\_EL2}$ or $\text{VTTBR\_EL2}$. VMID.

<table>
<thead>
<tr>
<th>SC2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample $\text{VTTBR_EL2}.\text{VMID}$.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample $\text{CONTEXTIDR_EL2}$.</td>
</tr>
</tbody>
</table>
On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

**NS, bit [18]**

Non-secure status. When in Debug state, gives the current Security state:

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure state, IsSecure() == TRUE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure state, IsSecure() == FALSE.</td>
</tr>
</tbody>
</table>

In Non-debug state, this bit is UNKNOWN.

Access to this field is RO.

**Bit [17]**

Reserved, RES0.

**SDD, bit [16]**

Secure debug disabled.

On entry to Debug state:

- If entering in Secure state, SDD is set to 0.
- If entering in Non-secure state, SDD is set to the inverse of ExternalSecureInvasiveDebugEnabled().

In Debug state, the value of the SDD bit does not change, even if ExternalSecureInvasiveDebugEnabled() changes.

In Non-debug state:

- SDD returns the inverse of ExternalSecureInvasiveDebugEnabled(). If the authentication signals that control ExternalSecureInvasiveDebugEnabled() change, a context synchronization event is required to guarantee their effect.
- This bit is unaffected by the Security state of the PE.

If EL3 is not implemented and the implementation is Non-secure, this bit is RES1.

Access to this field is RO.

**Bit [15]**

Reserved, RES0.

**HDE, bit [14]**

Halting debug enable. The possible values of this field are:

<table>
<thead>
<tr>
<th>HDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Halting disabled for Breakpoint, Watchpoint and Halt Instruction debug events.</td>
</tr>
<tr>
<td>0b1</td>
<td>Halting enabled for Breakpoint, Watchpoint and Halt Instruction debug events.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.
RW, bits [13:10]

Exception level Execution state status. In Debug state, each bit gives the current Execution state of each Exception level:

<table>
<thead>
<tr>
<th>RW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>All Exception levels are using AArch64 or the PE is in Non-debug state.</td>
</tr>
<tr>
<td>0b1110</td>
<td>The PE is in Debug state. EL0 is using AArch32. All other Exception levels are using AArch64. Only permitted if the PE is executing at EL0.</td>
</tr>
<tr>
<td>0b110x</td>
<td>The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 and EL3 are using AArch64. Only permitted if EL2 is implemented and enabled in the current Security state.</td>
</tr>
<tr>
<td>0b10xx</td>
<td>The PE is in Debug state. EL0, EL1, and, if implemented in the current Security state, EL2 are using AArch32. EL3 is using AArch64. Only permitted if EL3 is implemented.</td>
</tr>
<tr>
<td>0b0xxx</td>
<td>The PE is in Debug state. All Exception levels are using AArch32.</td>
</tr>
</tbody>
</table>

In Non-debug state, this field is RAO.

Access to this field is RO.

EL, bits [9:8]

Exception level. In Debug state, this gives the current EL of the PE.

In Non-debug state, this field is RAZ.

Access to this field is RO.

A, bit [7]

SEError interrupt pending. In Debug state, indicates whether an SError interrupt is pending:

- If \texttt{HCR.EL2.\{AMO, TGE\} = \{1, 0\}}, EL2 is enabled in the current Security state, and the PE is executing at EL0 or EL1, a virtual SError interrupt.
- Otherwise, a physical SError interrupt.

<table>
<thead>
<tr>
<th>A</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No SError interrupt pending.</td>
</tr>
<tr>
<td>0b1</td>
<td>SError interrupt pending.</td>
</tr>
</tbody>
</table>

A debugger can read EDSCR to check whether an SError interrupt is pending without having to execute further instructions. A pending SError might indicate data from target memory is corrupted.

UNKNOWN in Non-debug state.

Access to this field is RO.

ERR, bit [6]

Cumulative error flag. This bit is set to 1 following exceptions in Debug state and on any signaled overrun or underrun on the DTR or EDITR.

On a Cold reset, this field resets to 0.

Access to this field is RO.

STATUS, bits [5:0]

Debug status flags.
<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000001</td>
<td>PE is restarting, exiting Debug state.</td>
</tr>
<tr>
<td>0b000010</td>
<td>PE is in Non-debug state.</td>
</tr>
<tr>
<td>0b000111</td>
<td>Breakpoint.</td>
</tr>
<tr>
<td>0b010011</td>
<td>External debug request.</td>
</tr>
<tr>
<td>0b011011</td>
<td>Halting step, normal.</td>
</tr>
<tr>
<td>0b011111</td>
<td>Halting step, exclusive.</td>
</tr>
<tr>
<td>0b100011</td>
<td>OS Unlock Catch.</td>
</tr>
<tr>
<td>0b100111</td>
<td>Reset Catch.</td>
</tr>
<tr>
<td>0b101011</td>
<td>Watchpoint.</td>
</tr>
<tr>
<td>0b101111</td>
<td>HLT instruction.</td>
</tr>
<tr>
<td>0b110011</td>
<td>Software access to debug register.</td>
</tr>
<tr>
<td>0b110111</td>
<td>Exception Catch.</td>
</tr>
<tr>
<td>0b111011</td>
<td>Halting step, no syndrome.</td>
</tr>
</tbody>
</table>

All other values of STATUS are reserved.

Access to this field is **RO**.

**Accessing the EDSCR**

**EDSCR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x088</td>
<td>EDSCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
EDVIDSR, External Debug Virtual Context Sample Register

The EDVIDSR characteristics are:

**Purpose**

Contains sampled values captured on reading EDPCSR[31:0].

**Configuration**

EDVIDSR is in the Core power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.0-PCSample is implemented and ARMv8.2-PCSample is not implemented. Otherwise, direct accesses to EDVIDSR are **RES0**.

Implemented only if the **OPTIONAL** PC Sample-based Profiling Extension is implemented in the external debug registers space.

When the PC Sample-based Profiling Extension is implemented in the external debug registers space, if EL2 is not implemented and EL3 is not implemented, it is **IMPLEMENTATION DEFINED** whether EDVIDSR is implemented.

---

**Note**

ARMv8.2-PCSample implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

---

**Attributes**

If ARMv8.1-VHE is implemented, the format of this register differs depending on the value of **EDSCR.SC2**.

**Field descriptions**

The EDVIDSR bit assignments are:

**When ARMv8.1-VHE is not implemented or EDSCR.SC2 == 0:**

<table>
<thead>
<tr>
<th>NS</th>
<th>E2</th>
<th>E3</th>
<th>HV</th>
<th>RES0</th>
<th>VMID[15:8]</th>
<th>VMID</th>
</tr>
</thead>
</table>

This format applies in all Armv8.0 implementations.

**NS, bit [31]**

Non-secure state sample. Indicates the Security state associated with the most recent EDPCSR sample.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample is from Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample is from Non-secure state.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**E2, bit [30]**
When HaveEL(EL2):

Exception level 2 status sample. Indicates whether the most recent EDPCSR sample was associated with EL2.

<table>
<thead>
<tr>
<th>E2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample is not from EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample is from EL2.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3, bit [29]

When HaveEL(EL3) and !HighestELUsingAArch32():

Exception level 3 status sample. Indicates whether the most recent EDPCSR sample was associated with EL3 using AArch64.

<table>
<thead>
<tr>
<th>E3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample is not from EL3 using AArch64.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample is from EL3 using AArch64.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HV, bit [28]

EDPCSRhi (EDPCSR[63:32]) valid. Indicates whether bits [63:32] of the most recent EDPCSR sample might be nonzero:

<table>
<thead>
<tr>
<th>HV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Bits[63:32] of the most recent EDPCSR sample are zero.</td>
</tr>
<tr>
<td>0b1</td>
<td>Bits[63:32] of the most recent EDPCSR sample might be nonzero.</td>
</tr>
</tbody>
</table>

An EDVIDSR.HV value of 1 does not mean that the value of EDPCSRhi is nonzero. An EDVIDSR.HV value of 0 is a hint that EDPCSRhi (EDPCSR[63:32]) does not need to be read.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [27:16]

Reserved, RES0.

VMID[15:8], bits [15:8]

When ARMv8.1-VMID16 is implemented and HaveEL(EL2):

Extension to VMID[7:0]. See VMID[7:0] for more details.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.
VMID, bits [7:0]

When HaveEL(EL2):

VMID sample. The VMID associated with the most recent EDPCSRlo (EDPCSR[31:0]) sample. When the most recent EDPCSR sample was generated:

- This field is RES0 if any of the following apply:
  - The PE is executing in Secure state.
  - The PE is executing at EL2.
- Otherwise:
  - If EL2 is using AArch64 and either ARMv8.1-VMID16 is not implemented or VTCR_EL2.VS is 1, this field is set to VTTBR_EL2.VMID.
  - If EL2 is using AArch64, ARMv8.1-VMID16 is implemented, and VTCR_EL2.VS is 0, PMVIDSR.VMID[7:0] is set to VTTBR_EL2.VMID[7:0] and PMVIDSR.VMID[15:8] is RES0.
  - If EL2 is using AArch32, this field is set to VTTBR.VMID.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When ARMv8.1-VHE is implemented and EDSCR.SC2 == 1:

CONTEXTIDR_EL2, bits [31:0]

Context ID. The value of CONTEXTIDR_EL2 that is associated with the most recent EDPCSR sample. When the most recent EDPCSR sample was generated:

- If EL2 was using AArch64 and the PE was executing in Non-secure state, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- If EL2 was using AArch32 or the PE was executing in Secure state, then this field is set to an UNKNOWN value.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the EDVIDSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile

EDVIDSR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x0A8</td>
<td>EDVIDSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
EDWAR, External Debug Watchpoint Address Register

The EDWAR characteristics are:

Purpose

Returns the virtual data address being accessed when a Watchpoint Debug Event was triggered.

Configuration

EDWAR is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

Attributes

EDWAR is a 64-bit register.

Field descriptions

The EDWAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>62</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>59</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>56</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>53</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>50</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>49</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>47</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>46</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>44</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>43</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>41</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>40</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>39</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>38</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>36</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>35</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>33</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>32</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>31</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>30</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>29</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>28</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>27</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>26</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>25</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>24</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>23</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>22</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>21</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>20</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>19</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>18</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>17</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>16</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>15</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>14</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>13</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>12</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>11</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>10</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>9</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>8</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>7</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>6</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>5</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>4</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>3</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>2</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>1</td>
<td>Watchpoint address</td>
</tr>
<tr>
<td>0</td>
<td>Watchpoint address</td>
</tr>
</tbody>
</table>

Bits [63:0]

Watchpoint address. The data virtual address being accessed when a Watchpoint Debug Event was triggered and caused entry to Debug state. This address must be within a naturally-aligned block of memory of power-of-two size no larger than the DC ZVA block size.

The value of this register is UNKNOWN if the PE is in Non-debug state, or if Debug state was entered other than for a Watchpoint debug event.

The value of EDWAR[63:32] is UNKNOWN if Debug state was entered for a Watchpoint debug event taken from AArch32 state.

The EDWAR is subject to the same alignment rules as the reporting of a watchpointed address in the FAR. See ‘Determining the memory location that caused a Watchpoint exception’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D2 (AArch64 Self-hosted Debug)

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the EDWAR

EDWAR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x030</td>
<td>EDWAR</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x034</td>
<td>EDWAR</td>
<td>63:32</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

**Purpose**

Provides discovery information for the component.

**Configuration**

Implementation of this register is **OPTIONAL**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRCIDR0 are **RES0**.

**Attributes**

ERRCIDR0 is a 32-bit register.

**Field descriptions**

The ERRCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>0</td>
<td>PRMBL_0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, **RES0**.

**PRMBL_0, bits [7:0]**

Component identification preamble, segment 0. This field reads as **0x0D**.

**Accessing the ERRCIDR0**

ERRCIDR0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFF0</td>
<td>ERRCIDR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The ERRCIDR1 characteristics are:

**Purpose**

Provides discovery information for the component.

**Configuration**

Implementation of this register is **optional**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRCIDR1 are **RES0**.

**Attributes**

ERRCIDR1 is a 32-bit register.

**Field descriptions**

The ERRCIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7-4</td>
<td>Component class</td>
</tr>
<tr>
<td>3-0</td>
<td>Component identification preamble, segment 1</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**CLASS, bits [7:4]**

Component class.

<table>
<thead>
<tr>
<th>CLASS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>Generic peripheral with IMPLEMENTATION DEFINED register layout.</td>
</tr>
</tbody>
</table>

This field reads as 0xF. Other values are defined by the CoreSight Architecture.

**PRMBL_1, bits [3:0]**

Component identification preamble, segment 1. This field reads as 0x0.

**Accessing the ERRCIDR1**

**ERRCIDR1 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFF4</td>
<td>ERRCIDR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

**Purpose**

Provides discovery information for the component.

**Configuration**

Implementation of this register is **OPTIONAL**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRCIDR2 are **RES0**.

**Attributes**

ERRCIDR2 is a 32-bit register.

**Field descriptions**

The ERRCIDR2 bit assignments are:

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| RES0 | PRMBL_2 |

**Bits [31:8]**

Reserved, **RES0**.

**PRMBL_2, bits [7:0]**

Component identification preamble, segment 2. This field reads as **0x05**.

**Accessing the ERRCIDR2**

ERRCIDR2 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFF8</td>
<td>ERRCIDR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

**Purpose**

Provides discovery information for the component.

**Configuration**

Implementation of this register is **OPTIONAL**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRCIDR3 are **RES0**.

**Attributes**

ERRCIDR3 is a 32-bit register.

**Field descriptions**

The ERRCIDR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | PRMBL_3 |

**Bits [31:8]**

Reserved, **RES0**.

**PRMBL_3, bits [7:0]**

Component identification preamble, segment 3. This field reads as 0xB1.

**Accessing the ERRCIDR3**

ERRCIDR3 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFFC</td>
<td>ERRCIDR3</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The ERRCRICR0 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRCRICR0 is architecturally mapped to External register ERRIRQCR4.

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERRCRICR0 are **RES0**.

Present only if interrupt configuration registers use the recommended format. Otherwise, this register is **RES0**.

**Attributes**

ERRCRICR0 is a 64-bit register.

**Field descriptions**

The ERRCRICR0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| **RES0** | **ADDR** | **RES0** |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |

**Bits [63:56]**

Reserved, **RES0**.

**ADDR, bits [55:2]**

Message Signaled Interrupt address.

Specifies the address that the component writes to when signaling an interrupt.

The size of a physical address is **IMPLEMENTATION DEFINED**. Unimplemented high-order physical address bits are **RES0**.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally **UNKNOWN** value.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [1:0]**

Reserved, **RES0**.
Accessing the **ERRCRICR0**

**ERRCRICR0** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xEA0</td>
<td>ERRCRICR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
ERRCRICR1, Critical Error Interrupt Configuration Register 1

The ERRCRICR1 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRCRICR1 bits [31:0] are architecturally mapped to External register `ERRIRQCR5[31:0]`.

RW fields in this register reset to architecturally `UNKNOWN` values.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERRCRICR1 are `RES0`.

Present only if interrupt configuration registers use the recommended format. Otherwise, this register is `RES0`.

**Attributes**

ERRCRICR1 is a 32-bit register.

**Field descriptions**

The ERRCRICR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>DATA</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**DATA, bits [31:0]**

Payload for a message signaled interrupt.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally `UNKNOWN` value.
- On a Cold reset, this field resets to an architecturally `UNKNOWN` value.

**Accessing the ERRCRICR1**

ERRCRICR1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xEA8</td>
<td>ERRCRICR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are `RW`.
The ERRCRICR2 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRCRICR2 bits [31:0] are architecturally mapped to External register ERRIRQCR5[63:32]. Some or all RW fields of this register have defined reset values. This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERRCRICR2 are RES0. Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRCRICR2 is a 32-bit register.

**Field descriptions**

The ERRCRICR2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | IRQEN | NSMSI | SH | MemAttr |

**Bits [31:8]**

Reserved, RES0.

**IRQEN, bit [7]**

Message Signaled Interrupt enable.

Enables generation of message signalled interrupts.

<table>
<thead>
<tr>
<th>IRQEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Message signalled interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Message signalled interrupts are enabled.</td>
</tr>
</tbody>
</table>

If the component does not support disabling message signalled interrupts, this bit is RES0.

The following resets apply:

- On a Error recovery reset, this field resets to 0.
- On a Cold reset, this field resets to 0.

**NSMSI, bit [6]**

Security attribute.

Defines the physical address space for message signalled interrupts.
Physical address space for message signaled interrupts is Secure.

If the component prohibits Non-secure writes and does not support configuring the Security attribute, then the Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

If the component allows Non-secure writes, then the Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0 if any of the following are true:

- The component allows Non-secure writes.
- The component does not support configuring the Security attribute.

On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

**SH, bits [5:4]**

Shareability.

Defines the Shareability domain for message signaled interrupts.

<table>
<thead>
<tr>
<th>SH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Message signaled interrupts are in the Not shared Shareability domain.</td>
</tr>
<tr>
<td>0b10</td>
<td>Message signaled interrupts are in the Outer Shareable Shareability domain.</td>
</tr>
<tr>
<td>0b11</td>
<td>Message signaled interrupts are in the Inner Shareable Shareability domain.</td>
</tr>
</tbody>
</table>

If the component does not support configuring the Shareability domain, this field is RES0, meaning the Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**MemAttr, bits [3:0]**

Memory type.

Defines the memory type for message signaled interrupts. The values which correspond to each memory type are:

<table>
<thead>
<tr>
<th>MemAttr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Device-nGnRE.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Device-nGRE.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Device-GRE.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Outer Non-cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Outer Non-cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Outer Non-cacheable, Inner Write-Back Cacheable.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Outer Write-Through Cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Outer Write-Through Cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Outer Write-Through Cacheable, Inner Write-Back Cacheable.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Outer Write-Back Cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Outer Write-Back Cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Outer Write-Back Cacheable, Inner Write-Back Cacheable.</td>
</tr>
</tbody>
</table>
If the component does not support configuring the memory type, this field is RES0, meaning the memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the ERRCRICR2

ERRCRICR2 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xEAC</td>
<td>ERRCRICR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
The ERRDEVAFF characteristics are:

**Purpose**

For a group that has affinity with a single PE or cluster of PEs, ERRDEVAFF is a copy of MPIDR_EL1 or part of MPIDR_EL1:

- If the group of error records has affinity with a single PE, the affinity level is 0, ERRDEVAFF reads the same value as MPIDR_EL1, and ERRDEVAFF.F0V reads-as-one to indicate affinity level 0.
- If the group of error records has affinity with a cluster of PEs, the affinity level is 1, 2, or 3, parts of ERRDEVAFF reads the same value as parts of MPIDR_EL1, and the rest of ERRDEVAFF indicates the level.

For example, if the affinity level is 2 then all of the following are true:

- The group of error records has affinity with all PEs in the system that have the same value in MPIDR_EL1.\{Aff3,Aff2\}.
- ERRDEVAFF.\{Aff3,Aff2\} reads the same value as MPIDR_EL1.\{Aff3,Aff2\}.
- ERRDEVAFF.Aff1 reads as 0x80 and ERRDEVAFF.\{Aff0,F0V\} read-as-zero, to indicate affinity level 2.

**Configuration**

Implementation of this register is OPTIONAL.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRDEVAFF are RES0. Present only if a group has affinity with a PE or cluster of PEs, and otherwise RES0.

**Attributes**

ERRDEVAFF is a 64-bit register.

**Field descriptions**

The ERRDEVAFF bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    | RES0 |    |    | Aff3 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:40]**

Reserved, RES0.

**Aff3, bits [39:32]**

Affinity level 3. The MPIDR_EL1.Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

**F0V, bit [31]**

Indicates that the ERRDEVAFF.Aff0 field is valid.

<table>
<thead>
<tr>
<th>F0V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERRDEVAFF.Aff0 is not valid, and the PE affinity level is 1, 2 or 3.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERRDEVAFF.Aff0 is valid, and the PE affinity level is 0.</td>
</tr>
</tbody>
</table>
U, bit [30]

Uniprocessor. The MPIDR_EL1.U bit viewed from the highest Exception level of the associated PE.

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The PE is part of a multiprocessor system.</td>
</tr>
<tr>
<td>1</td>
<td>The PE is part of a uniprocessor system.</td>
</tr>
</tbody>
</table>

If ERRDEVAFF.Aff0 is not valid, this bit is not valid and reads as **UNKNOWN**.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Multithreaded. The MPIDR_EL1.MT bit viewed from the highest Exception level of the associated PE.

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Performance of PEs at the lowest affinity level is largely independent.</td>
</tr>
<tr>
<td>1</td>
<td>Performance of PEs at the lowest affinity level is very interdependent.</td>
</tr>
</tbody>
</table>

If ERRDEVAFF.Aff0 is not valid, this bit is not valid and reads as **UNKNOWN**.

Aff2, bits [23:16]

Affinity level 2.

When the PE affinity level is 0, 1, or 2, this field is the MPIDR_EL1.Aff2 field viewed from the highest Exception level of the associated PE or PEs.

When the PE affinity level is 3, this field indicates that ERRDEVAFF.Aff3 field is valid, and the following values are defined:

<table>
<thead>
<tr>
<th>Aff2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x80</td>
<td>ERRDEVAFF.Aff3 is valid, and the PE affinity level is 3.</td>
</tr>
</tbody>
</table>

This field reads as 0x80.

All other values are reserved.

Aff1, bits [15:8]

Affinity level 1.

When the PE affinity level is 0 or 1, this field is the MPIDR_EL1.Aff1 field viewed from the highest Exception level of the associated PE or PEs.

When the PE affinity level is 2 or 3, this field indicates that ERRDEVAFF.Aff2 is valid, and the following values are defined:

<table>
<thead>
<tr>
<th>Aff1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>ERRDEVAFF.Aff2 is not valid, and the PE affinity level is 3.</td>
</tr>
<tr>
<td>0x80</td>
<td>ERRDEVAFF.Aff2 is valid, and the PE affinity level is 2.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Aff0, bits [7:0]

Affinity level 0.

When the PE affinity level is 0, this field is the MPIDR_EL1.Aff0 field viewed from the highest Exception level of the associated PE or PEs.
When the PE affinity level is 1, 2 or 3, this field indicates that ERRDEVAFF.Aff1 is valid, and the following values are defined:

<table>
<thead>
<tr>
<th>Aff0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>ERRDEVAFF.Aff1 is not valid, and the PE affinity level is 2 or 3.</td>
</tr>
<tr>
<td>0x80</td>
<td>ERRDEVAFF.Aff1 is valid, and the PE affinity level is 1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Accessing the ERRDEVAFF**

**ERRDEVAFF can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFA8</td>
<td>ERRDEVAFF</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

**Purpose**

Provides discovery information for the component.

**Configuration**

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRDEVARCH are **UNDEFINED**.

**Attributes**

ERRDEVARCH is a 32-bit register.

**Field descriptions**

The ERRDEVARCH bit assignments are:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| ARCHITECT | PRESENT | REVISION | ARCHVER | ARCHPART |

**ARCHITECT, bits [31:21]**

Architect.

Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

<table>
<thead>
<tr>
<th>ARCHITECT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01000111011</td>
<td>JEP106 continuation code 0x4, ID code 0x3B. Arm Limited.</td>
</tr>
</tbody>
</table>

This field reads as 0x23B.

Other values are defined by the JEDEC JEP106 standard.

**PRESENT, bit [20]**

DEVARCH Present.

Defines that the DEVARCH register is present.

<table>
<thead>
<tr>
<th>PRESENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Device Architecture information not present.</td>
</tr>
<tr>
<td>0b1</td>
<td>Device Architecture information present.</td>
</tr>
</tbody>
</table>

This bit is RAO.

**REVISION, bits [19:16]**

Revision.

Defines the architecture revision of the component. The defined values of this field are:
<table>
<thead>
<tr>
<th>REVISION</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>RAS System Architecture v1.0.</td>
</tr>
<tr>
<td>0b0001</td>
<td>RAS System Architecture v1.1. As 0b0000 and also:</td>
</tr>
<tr>
<td></td>
<td>• Simplifies ERR&lt;n&gt;STATUS.</td>
</tr>
<tr>
<td></td>
<td>• Adds support for additional ERR&lt;n&gt;MISC&lt;m&gt; registers.</td>
</tr>
<tr>
<td></td>
<td>• Adds support for the optional RAS Timestamp Extension.</td>
</tr>
<tr>
<td></td>
<td>• Adds support for the optional RAS Common Fault Injection Model Extension.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**ARCHVER, bits [15:12]**

Architecture Version.

Defines the architecture version of the component. The defined values of this field are:

<table>
<thead>
<tr>
<th>ARCHVER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>RAS System Architecture v1.</td>
</tr>
</tbody>
</table>

This field reads as 0b0000.

All other values are reserved.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

**ARCHPART, bits [11:0]**

Architecture Part.

Defines the architecture of the component.

<table>
<thead>
<tr>
<th>ARCHPART</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA00</td>
<td>RAS system architecture.</td>
</tr>
</tbody>
</table>

This register reads as 0xA00.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

**Accessing the ERRDEVARCH**

**ERRDEVARCH can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFBC</td>
<td>ERRDEVARCH</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
ERRDEVID, Device Configuration Register

The ERRDEVID characteristics are:

**Purpose**

Provides discovery information for the component.

**Configuration**

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRDEVID are UNDEFINED.

**Attributes**

ERRDEVID is a 32-bit register.

**Field descriptions**

The ERRDEVID bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30-16</td>
<td>NUM</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**NUM, bits [15:0]**

Highest numbered index of the error records in this group, plus one. Each implemented record is owned by a node. A node might own multiple records.

This manual describes the memory-mapped view of a group with up to 56 records, the most that can be contained in a 4KB component, meaning the highest possible value for this field is 56.

This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the ERRDEVID**

**ERRDEVID can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFC8</td>
<td>ERRDEVID</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRERICR0, Error Recovery Interrupt Configuration Register 0

The ERRERICR0 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRERICR0 is architecturally mapped to External register ERRIRQCR2. RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRERICR0 are RES0. Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRERICR0 is a 64-bit register.

**Field descriptions**

The ERRERICR0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 | ADDR | RES0 |

**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:2]**

Message Signaled Interrupt address.

Specifies the address that the component writes to when signaling an interrupt.

The size of a physical address is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [1:0]**

Reserved, RES0.
**Accessing the ERRERICR0**

**ERRERICR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE90</td>
<td>ERRERICR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
ERRERICR1, Error Recovery Interrupt Configuration Register 1

The ERRERICR1 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRERICR1 bits [31:0] are architecturally mapped to External register ERRIRQCR3[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRERICR1 are RES0.

Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRERICR1 is a 32-bit register.

**Field descriptions**

The ERRERICR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | DATA |

**DATA, bits [31:0]**

Payload for a message signaled interrupt.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the ERRERICR1**

**ERRERICR1 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE98</td>
<td>ERRERICR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERRERICR2, Error Recovery Interrupt Configuration Register 2

The ERRERICR2 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRERICR2 bits [31:0] are architecturally mapped to External register ERRIRQCR3[63:32]. Some or all RW fields of this register have defined reset values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRERICR2 are RES0. Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRERICR2 is a 32-bit register.

**Field descriptions**

The ERRERICR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>IRQEN</td>
<td>Message Signaled Interrupt enable. Enables generation of message signaled interrupts.</td>
</tr>
<tr>
<td>29</td>
<td>NSMSI</td>
<td>Security attribute. Defines the physical address space for message signaled interrupts.</td>
</tr>
<tr>
<td>28</td>
<td>SH</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>MemAttr</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**IRQEN, bit [7]**

Message Signaled Interrupt enable.

Enables generation of message signaled interrupts.

If the component does not support disabling message signaled interrupts, this bit is RES0.

The following resets apply:

- On a Error recovery reset, this field resets to 0.
- On a Cold reset, this field resets to 0.
### NSMSI

<table>
<thead>
<tr>
<th>NSMSI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Physical address space for message signaled interrupts is Secure.</td>
</tr>
<tr>
<td>0b1</td>
<td>Physical address space for message signaled interrupts is Non-secure.</td>
</tr>
</tbody>
</table>

If the component prohibits Non-secure writes and does not support configuring the Security attribute, then the Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

If the component allows Non-secure writes, then the Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0 if any of the following are true:

- The component allows Non-secure writes.
- The component does not support configuring the Security attribute.

On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

### SH, bits [5:4]

Shareability.

Defines the Shareability domain for message signaled interrupts.

<table>
<thead>
<tr>
<th>SH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Message signaled interrupts are in the Not shared Shareability domain.</td>
</tr>
<tr>
<td>0b10</td>
<td>Message signaled interrupts are in the Outer Shareable Shareability domain.</td>
</tr>
<tr>
<td>0b11</td>
<td>Message signaled interrupts are in the Inner Shareable Shareability domain.</td>
</tr>
</tbody>
</table>

If the component does not support configuring the Shareability domain, this field is RES0, meaning the Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

The following resets apply:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

### MemAttr, bits [3:0]

Memory type.

Defines the memory type for message signaled interrupts. The values which correspond to each memory type are:

<table>
<thead>
<tr>
<th>MemAttr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Device-nGnRE.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Device-nGRE.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Device-GRE.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Outer Non-cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Outer Non-cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Outer Non-cacheable, Inner Write-Back Cacheable.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Outer Write-Through Cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Outer Write-Through Cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Outer Write-Through Cacheable, Inner Write-Back Cacheable.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Outer Write-Back Cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Outer Write-Back Cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Outer Write-Back Cacheable, Inner Write-Back Cacheable.</td>
</tr>
</tbody>
</table>
If the component does not support configuring the memory type, this field is RES0, meaning the memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

**Note**

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the ERRERICR2**

**ERRERICR2 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE9C</td>
<td>ERRERICR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERRFHICR0, Fault-Handling Interrupt Configuration Register 0

The ERRFHICR0 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRFHICR0 is architecturally mapped to External register ERRIRQCR0. RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRFHICR0 are RES0. Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRFHICR0 is a 64-bit register.

**Field descriptions**

The ERRFHICR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>[63:56]</td>
<td>Reserved, RES0</td>
<td>0</td>
</tr>
<tr>
<td>[1:0]</td>
<td>Reserved, RES0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:56]**

Reserved, RES0.

**ADDR, bits [55:2]**

Message Signaled Interrupt address.

Specifies the address that the component writes to when signaling an interrupt.

The size of a physical address is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [1:0]**

Reserved, RES0.
Accessing the ERRFHIRCR0

ERRFHIRCR0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE80</td>
<td>ERRFHIRCR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
The ERRFHICR1 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRFHICR1 bits [31:0] are architecturally mapped to External register ERRIRQCR1[31:0]. RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRFHICR1 are RES0.

Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRFHICR1 is a 32-bit register.

**Field descriptions**

The ERRFHICR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| DATA |

**DATA, bits [31:0]**

Payload for a message signaled interrupt.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the ERRFHICR1**

ERRFHICR1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE88</td>
<td>ERRFHICR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERRFHICR2, Fault-Handling Interrupt Configuration Register 2

The ERRFHICR2 characteristics are:

**Purpose**

Interrupt configuration register.

**Configuration**

External register ERRFHICR2 bits [31:0] are architecturally mapped to External register ERRIRQCR1[63:32]. Some or all RW fields of this register have defined reset values. This register is present only when RAS is implemented. Otherwise, direct accesses to ERRFHICR2 are RES0. Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRFHICR2 is a 32-bit register.

**Field descriptions**

The ERRFHICR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
<td>RES0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>IRQEN</td>
<td></td>
<td>Message Signaled Interrupt enable. Enables generation of message signaled interrupts.</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>NSMSI</td>
<td></td>
<td>Security attribute. Defines the physical address space for message signaled interrupts.</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If the component does not support disabling message signaled interrupts, this bit is RES0.

The following resets apply:

- On a Error recovery reset, this field resets to 0.
- On a Cold reset, this field resets to 0.
<table>
<thead>
<tr>
<th>NSMSI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Physical address space for message signaled interrupts is Secure.</td>
</tr>
<tr>
<td>0b1</td>
<td>Physical address space for message signaled interrupts is Non-secure.</td>
</tr>
</tbody>
</table>

If the component prohibits Non-secure writes and does not support configuring the Security attribute, then the Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

If the component allows Non-secure writes, then the Security attribute used for message signaled interrupts is Non-secure.

This bit is RES0 if any of the following are true:

- The component allows Non-secure writes.
- The component does not support configuring the Security attribute.

On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

**SH, bits [5:4]**

Shareability.

Defines the Shareability domain for message signaled interrupts.

<table>
<thead>
<tr>
<th>SH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Message signaled interrupts are in the Not shared Shareability domain.</td>
</tr>
<tr>
<td>0b10</td>
<td>Message signaled interrupts are in the Outer Shareable Shareability domain.</td>
</tr>
<tr>
<td>0b11</td>
<td>Message signaled interrupts are in the Inner Shareable Shareability domain.</td>
</tr>
</tbody>
</table>

If the component does not support configuring the Shareability domain, this field is RES0, meaning the Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**MemAttr, bits [3:0]**

Memory type.

Defines the memory type for message signaled interrupts. The values which correspond to each memory type are:

<table>
<thead>
<tr>
<th>MemAttr</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b0001</td>
<td>Device-nGnRE.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Device-nGRE.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Device-GRE.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Outer Non-cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Outer Non-cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Outer Non-cacheable, Inner Write-Back Cacheable.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Outer Write-Through Cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Outer Write-Through Cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Outer Write-Through Cacheable, Inner Write-Back Cacheable.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Outer Write-Back Cacheable, Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Outer Write-Back Cacheable, Inner Write-Through Cacheable.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Outer Write-Back Cacheable, Inner Write-Back Cacheable.</td>
</tr>
</tbody>
</table>
If the component does not support configuring the memory type, this field is RES0, meaning the memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

**Note**

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the ERRFHIRCR2**

ERRFHIRCR2 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE8C</td>
<td>ERRFHIRCR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERRGSR, Error Group Status Register

The ERRGSR characteristics are:

**Purpose**

ERRGSR shows the status for the records in the group.

**Configuration**

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRGSR are UNDEFINED.

This manual describes the memory-mapped view of a group with up to 56 records, the most that can be contained in a 4KB component. Extra records might be added by increasing the page size and extending ERRGSR into multiple registers.

**Attributes**

ERRGSR is a 64-bit register.

**Field descriptions**

The ERRGSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>m</td>
<td>S&lt;m&gt;, bit [m], for m = 0 to 55</td>
</tr>
</tbody>
</table>

The status for Error Record <m>. A read-only copy of `ERR<m>STATUS` V.

If the corresponding record is not implemented, or the corresponding record does not support this type of reporting, this bit is RES0.

**Accessing the ERRGSR**

ERRGSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE00</td>
<td>ERRGSR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRIIDR, Implementation Identification Register

The ERRIIDR characteristics are:

**Purpose**

Defines the implementer of the component.

**Configuration**

Implementation of this register is **OPTIONAL**.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERRIIDR are **RES0**.

**Attributes**

ERRIIDR is a 32-bit register.

**Field descriptions**

The ERRIIDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProductID</strong></td>
<td><strong>Variant</strong></td>
<td><strong>Revision</strong></td>
<td><strong>Implementer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ProductID, bits [31:20]**

IMPLEMENTATION **DEFINED**.

Part number, bits [11:0]. The part number is selected by the designer of the component.

If ERRIIDR0 and ERRIIDR1 are implemented, ERRIIDR0.PART_0 matches bits [7:0] of ERRIIDR.ProductID and ERRIIDR1.PART_1 matches bits [11:8] of ERRIIDR.ProductID.

**Variant, bits [19:16]**

IMPLEMENTATION **DEFINED**.

Component major revision.

This field distinguishes product variants or major revisions of the product.

If ERRIIDR2 is implemented, ERRIIDR2.REVISION matches ERRIIDR.Variant.

**Revision, bits [15:12]**

IMPLEMENTATION **DEFINED**.

Component minor revision.

This field distinguishes minor revisions of the product.

If ERRIIDR3 is implemented, ERRIIDR3.REVAND matches ERRIIDR.Revision.

**Implementer, bits [11:0]**

IMPLEMENTATION **DEFINED**.
Contains the JEP106 code of the company that implemented the RAS component. For an Arm implementation, this field has the value 0x43B.

Bits [11:8] contain the JEP106 continuation code of the implementer, and bits [6:0] contain the JEP106 identity code of the implementer. Bit 7 is RES0.

If ERRPIDR4 is implemented, ERRPIDR2 is implemented, and ERRPIDR1 is implemented, ERRPIDR4.DES_2 matches bits [11:8] of ERRIIDR. Implementer, ERRPIDR2.DES_1 matches bits [6:4] of ERRIIDR. Implementer, and ERRPIDR1.DES_0 matches bits [3:0] of ERRIIDR. Implementer.

Accessing the ERRIIDR

**ERRIIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE10</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRIRQCR\(n\), Generic Error Interrupt Configuration Register, \(n = 0 - 15\)

The ERRIRQCR\(n\) characteristics are:

**Purpose**

The ERRIRQCR\(n\) registers are IMPLEMENTATION DEFINED interrupt configuration registers.

The architecture provides a recommended format for the ERRIRQCR\(n\) registers. The registers provided by the recommended layout are:

- **ERRFHICR0, ERRFHICR1, and ERRFHICR2**, the fault-handling interrupt configuration registers. ERRFHICR\(m\) maps to ERRIRQCR0 and ERRIRQCR1.

- **ERRERICR0, ERRERICR1, and ERRERICR2**, the error recovery interrupt configuration registers. ERRERICR\(n\) maps to ERRIRQCR2 and ERRIRQCR3.

- If ARMv8.4-RAS is implemented, **ERRCRI0, ERRCRI1, and ERRCRI2**, the critical error interrupt configuration registers. ERRFHICR\(m\) maps to ERRIRQCR4 and ERRIRQCR5.

- **ERRIROSR**, the error interrupt status register. ERRIROSR maps to ERRIRQCR15.

This register describes the generic IMPLEMENTATION DEFINED format of the interrupt configuration registers, when the recommended layout is not used.

**Configuration**

Present only if the interrupt configuration registers are implemented. Otherwise, this register is RES0.

**Attributes**

ERRIRQCR\(n\) is a 64-bit register.

**Field descriptions**

The ERRIRQCR\(n\) bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED controls. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**Accessing the ERRIRQCR\(n\)**

ERRIRQCR\(n\) can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xE80 + 8n</td>
<td>ERRIRQCR(n)</td>
</tr>
</tbody>
</table>
Accesses on this interface are RW.
ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

**Purpose**

Interrupt status register.

**Configuration**

External register ERRIRQSR is architecturally mapped to External register ERRIRQCR15.

RW fields in this register reset to architecturally UNKNOWN values.

Present only if interrupt configuration registers use the recommended format. Otherwise, this register is RES0.

**Attributes**

ERRIRQSR is a 64-bit register.

**Field descriptions**

The ERRIRQSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>CRIERR</td>
</tr>
<tr>
<td>61</td>
<td>CRI</td>
</tr>
<tr>
<td>60</td>
<td>FHIERR</td>
</tr>
<tr>
<td>59</td>
<td>FHI</td>
</tr>
<tr>
<td>58</td>
<td>ERIERR</td>
</tr>
<tr>
<td>57</td>
<td>ERI</td>
</tr>
<tr>
<td>56</td>
<td>G43ERR</td>
</tr>
<tr>
<td>55</td>
<td>G42ERR</td>
</tr>
<tr>
<td>54</td>
<td>G41ERR</td>
</tr>
<tr>
<td>53</td>
<td>G40ERR</td>
</tr>
<tr>
<td>52</td>
<td>G39ERR</td>
</tr>
<tr>
<td>51</td>
<td>G38ERR</td>
</tr>
<tr>
<td>50</td>
<td>G37ERR</td>
</tr>
<tr>
<td>49</td>
<td>G36ERR</td>
</tr>
<tr>
<td>48</td>
<td>G35ERR</td>
</tr>
<tr>
<td>47</td>
<td>G34ERR</td>
</tr>
<tr>
<td>46</td>
<td>G33ERR</td>
</tr>
<tr>
<td>45</td>
<td>G32ERR</td>
</tr>
<tr>
<td>44</td>
<td>G31ERR</td>
</tr>
<tr>
<td>43</td>
<td>G30ERR</td>
</tr>
<tr>
<td>42</td>
<td>G29ERR</td>
</tr>
<tr>
<td>41</td>
<td>G28ERR</td>
</tr>
<tr>
<td>40</td>
<td>G27ERR</td>
</tr>
<tr>
<td>39</td>
<td>G26ERR</td>
</tr>
<tr>
<td>38</td>
<td>G25ERR</td>
</tr>
<tr>
<td>37</td>
<td>G24ERR</td>
</tr>
<tr>
<td>36</td>
<td>G23ERR</td>
</tr>
<tr>
<td>35</td>
<td>G22ERR</td>
</tr>
<tr>
<td>34</td>
<td>G21ERR</td>
</tr>
<tr>
<td>33</td>
<td>G20ERR</td>
</tr>
<tr>
<td>32</td>
<td>G19ERR</td>
</tr>
<tr>
<td>31</td>
<td>G18ERR</td>
</tr>
<tr>
<td>30</td>
<td>G17ERR</td>
</tr>
<tr>
<td>29</td>
<td>G16ERR</td>
</tr>
<tr>
<td>28</td>
<td>G15ERR</td>
</tr>
<tr>
<td>27</td>
<td>G14ERR</td>
</tr>
<tr>
<td>26</td>
<td>G13ERR</td>
</tr>
<tr>
<td>25</td>
<td>G12ERR</td>
</tr>
<tr>
<td>24</td>
<td>G11ERR</td>
</tr>
<tr>
<td>23</td>
<td>G10ERR</td>
</tr>
<tr>
<td>22</td>
<td>G9ERR</td>
</tr>
<tr>
<td>21</td>
<td>G8ERR</td>
</tr>
<tr>
<td>20</td>
<td>G7ERR</td>
</tr>
<tr>
<td>19</td>
<td>G6ERR</td>
</tr>
<tr>
<td>18</td>
<td>G5ERR</td>
</tr>
<tr>
<td>17</td>
<td>G4ERR</td>
</tr>
<tr>
<td>16</td>
<td>G3ERR</td>
</tr>
<tr>
<td>15</td>
<td>G2ERR</td>
</tr>
<tr>
<td>14</td>
<td>G1ERR</td>
</tr>
<tr>
<td>13</td>
<td>G0ERR</td>
</tr>
<tr>
<td>12</td>
<td>G0ERR</td>
</tr>
<tr>
<td>11</td>
<td>G1ERR</td>
</tr>
<tr>
<td>10</td>
<td>G2ERR</td>
</tr>
<tr>
<td>9</td>
<td>G3ERR</td>
</tr>
<tr>
<td>8</td>
<td>G4ERR</td>
</tr>
<tr>
<td>7</td>
<td>G5ERR</td>
</tr>
<tr>
<td>6</td>
<td>G6ERR</td>
</tr>
<tr>
<td>5</td>
<td>G7ERR</td>
</tr>
<tr>
<td>4</td>
<td>G8ERR</td>
</tr>
<tr>
<td>3</td>
<td>G9ERR</td>
</tr>
<tr>
<td>2</td>
<td>G10ERR</td>
</tr>
<tr>
<td>1</td>
<td>G11ERR</td>
</tr>
<tr>
<td>0</td>
<td>G12ERR</td>
</tr>
</tbody>
</table>

This bit is read/write-one-to-clear.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally UNKNOWN value.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**CRIERR, bit [5]**

Critical error interrupt error.

<table>
<thead>
<tr>
<th>CRIERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupt write has not returned an error since this bit was last cleared to 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt write has returned an error since this bit was last cleared to 0.</td>
</tr>
</tbody>
</table>

**CRI, bit [4]**

Critical error interrupt write in progress.

<table>
<thead>
<tr>
<th>CRI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupt write not in progress.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt write in progress.</td>
</tr>
</tbody>
</table>
Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

**Note**

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual \texttt{ERR<n>STATUS} registers.

---

**ERIERR, bit [3]**

Error recovery interrupt error.

<table>
<thead>
<tr>
<th>ERIERR</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupt write has not returned an error since this bit was last cleared to 0.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt write has returned an error since this bit was last cleared to 0.</td>
<td></td>
</tr>
</tbody>
</table>

This bit read/write-one-to-clear.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally \texttt{UNKNOWN} value.
- On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**ERI, bit [2]**

Error recovery interrupt write in progress.

<table>
<thead>
<tr>
<th>ERI</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupt write not in progress.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt write in progress.</td>
<td></td>
</tr>
</tbody>
</table>

Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

**Note**

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual \texttt{ERR<n>STATUS} registers.

---

**FHIERR, bit [1]**

Fault handling interrupt error.

<table>
<thead>
<tr>
<th>FHIERR</th>
<th>Meaning</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupt write has not returned an error since this bit was last cleared to 0.</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt write has returned an error since this bit was last cleared to 0.</td>
<td></td>
</tr>
</tbody>
</table>

This bit read/write-one-to-clear.

The following resets apply:

- On a Error recovery reset, this field resets to an architecturally \texttt{UNKNOWN} value.
• On a Cold reset, this field resets to an architecturally UNKNOWN value.

**FHI, bit [0]**

Fault handling interrupt write in progress.

<table>
<thead>
<tr>
<th>FHI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupt write not in progress.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupt write in progress.</td>
</tr>
</tbody>
</table>

Software must not disable an interrupt whilst the write is in progress.

This bit is read-only.

**Note**

This bit does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual **ERR<n>STATUS** registers.

**Accessing the ERRIRQSR**

ERRIRQSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xEF8</td>
<td>ERRIRQSR</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
ERR<\(n\)>ADDR, Error Record Address Register, \(n = 0 - 65534\)

The ERR<\(n\)>ADDR characteristics are:

**Purpose**

If an error has an associated address, then this must be written to the address register when the error is recorded. It is implementation defined how the recorded addresses map to the software-visible physical addresses.

Software might have to reconstruct the actual physical addresses using the identity of the node and knowledge of the system.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERR<\(n\)>ADDR are undefined.

The number of error records that are implemented is implementation defined.

If error record \(<n>\) is not implemented, ERR<\(n\)>ADDR is RES0.

ERR<\(q\)>FR describes the features implemented by the node that owns error record \(<n>\). \(<q>\) is the index of the first error record owned by the same node as error record \(<n>\). If the node owns a single record, then \(q = n\).

If the error record does not record the fault address, this register is RES0.

**Attributes**

ERR<\(n\)>ADDR is a 64-bit register.

**Field descriptions**

The ERR<\(n\)>ADDR bit assignments are:

<table>
<thead>
<tr>
<th>NS</th>
<th>SI</th>
<th>AI</th>
<th>VA</th>
<th>RES0</th>
<th>PADDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
<td>61</td>
<td>60</td>
<td>59</td>
<td>58</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
</tr>
</tbody>
</table>

**NS, bit [63]**

Non-secure attribute.

<table>
<thead>
<tr>
<th>NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address is Secure.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address is Non-secure.</td>
</tr>
</tbody>
</table>

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**SI, bit [62]**

Secure Incorrect.
Indicates whether the NS bit is valid.

<table>
<thead>
<tr>
<th>SI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The NS bit is correct. That is, it matches the programmers' view of the Non-secure attribute for this recorded location.</td>
</tr>
<tr>
<td>0b1</td>
<td>The NS bit might not be correct, and might not match the programmers' view of the Non-secure attribute for the recorded location.</td>
</tr>
</tbody>
</table>

It is **IMPLEMENTATION DEFINED** whether this bit is read-only or read/write.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**AI, bit [61]**

Address Incorrect.

Indicates whether the PADDR field is a valid physical address that is known to match the programmers' view of the physical address for the recorded location.

<table>
<thead>
<tr>
<th>AI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PADDR field is a valid physical address. That is, it matches the programmers' view of the physical address for the recorded location.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PADDR field might not be a valid physical address, and might not match the programmers' view of the physical address for the recorded location.</td>
</tr>
</tbody>
</table>

It is **IMPLEMENTATION DEFINED** whether this bit is read-only or read/write.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**VA, bit [60]**

Virtual Address.

Indicates whether the PADDR field is a virtual address.

<table>
<thead>
<tr>
<th>VA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PADDR field is not a virtual address.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PADDR field is a virtual address.</td>
</tr>
</tbody>
</table>

No context information is provided for the virtual address. When this bit is set to 1, ERR<n>ADDR.{NS,SI,VA} must read as {0,1,1}.

Support for this bit is optional. If this bit is not implemented and the PADDR field is a virtual address, then ERR<n>ADDR.{NS,SI,VA} must read as {0,1,1}.

It is **IMPLEMENTATION DEFINED** whether this bit is read-only or read/write.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [59:56]**

Reserved, **RES0**.
PADDR, bits [55:0]

Physical Address. Address of the recorded location.

If the physical address size implemented by this component is smaller than the size of this field, then high-order bits are unimplemented and either RES0 or have a fixed read-only IMPLEMENTATION DEFINED value.

Low-order address bits might also be unimplemented and RES0, for example, if the physical address is always aligned to the size of a protection granule.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing the ERR<n>ADDR

ERR<n>ADDR ignores writes if ERR<n>STATUS.AV is set to 1.

ERR<n>ADDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x018 + 64n</td>
<td>ERR&lt;n&gt;ADDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERR<n>CTRL, Error Record Control Register, n = 0 - 65534

The ERR<n>CTRL characteristics are:

**Purpose**

The error control register contains enable bits for the node that writes to this record, which:

- Enable error detection and correction.
- Enable an error recovery interrupt.
- Enable a fault handling interrupt.
- Enable error recovery reporting as a read or write error response.
- When ARMv.4-RAS is implemented, enable a critical error interrupt.

For each bit, if the selected node does not support the feature, then the bit is RES0. The definition of each record is IMPLEMENTATION DEFINED.

**Configuration**

Some or all RW fields of this register have defined reset values.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If error record <n> is not implemented, or error record <n> is not the first error record owned by the node, ERR<n>CTRL is RES0.

ERR<n>FR describes the features implemented by the node.

**Attributes**

ERR<n>CTRL is a 64-bit register.

**Field descriptions**

The ERR<n>CTRL bit assignments are:

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0| CIRES0| WDUIDUI| WCFI| CFI| WUE| WFI| WUI| UE| FI| UI| IMPLEMENTATION DEFINED| ED|
| 13102928272625242322212019181716151413| 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   | 32  | 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  |

**IMPLEMENTATION DEFINED, bits [63:32]**

IMPLEMENTATION DEFINED.

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**Bits [31:14]**

Reserved, RES0.
**CI, bit [13]**

When ERR<\n>FR.CI == 0b10:

Critical error interrupt enable.

When enabled, the critical error interrupt is generated for a critical error condition.

<table>
<thead>
<tr>
<th>CI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Critical error interrupt not generated for critical errors. Critical</td>
</tr>
<tr>
<td></td>
<td>errors are treated as Uncontained errors.</td>
</tr>
<tr>
<td>0b1</td>
<td>Critical error interrupt generated for critical errors.</td>
</tr>
</tbody>
</table>

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bit [12]**

Reserved, RES0.

**WDUI, bit [11]**

When ERR<\n>FR.DUI == 0b11:

Error recovery interrupt for deferred errors on writes enable.

When enabled, the error recovery interrupt is generated for all detected Deferred errors on writes.

<table>
<thead>
<tr>
<th>WDUI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error recovery interrupt not generated for deferred errors on writes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error recovery interrupt generated for deferred errors on writes.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**DUI, bit [10]**

When ERR<\n>FR.DUI == 0b10:

Error recovery interrupt for deferred errors enable. This control applies to errors arising from both reads and writes.

When enabled, an error recovery interrupt is generated for all detected Deferred errors.

<table>
<thead>
<tr>
<th>DUI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error recovery interrupt not generated for deferred errors.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error recovery interrupt generated for deferred errors.</td>
</tr>
</tbody>
</table>
The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**When ERR<n>FR.DUI == 0b11:**

When ERR<n>FR.DUI == 0b11, this field is named RDUI.

Error recovery interrupt for deferred errors on reads enable.

When enabled, the error recovery interrupt is generated for all detected Deferred errors on reads.

<table>
<thead>
<tr>
<th>RDUI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error recovery interrupt not generated for deferred errors on reads.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error recovery interrupt generated for deferred errors on reads.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**WCFI, bit [9]**

**When ERR<n>FR.CFI == 0b11:**

Fault handling interrupt for Corrected errors on writes enable.

When enabled:

- If the node implements Corrected error counters, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 1. For more information, see ERR<n>MISC0.
- Otherwise, the fault handling interrupt is also generated for all detected Corrected errors on writes.

<table>
<thead>
<tr>
<th>WCFI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault handling interrupt not generated for Corrected errors on writes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault handling interrupt generated for Corrected errors on writes.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
**CFI, bit [8]**

**When \( \text{ERR}<n>\text{FR.CFI} == 0b10: \)**

Fault handling interrupt for Corrected errors enable. This control applies to errors arising from both reads and writes.

When enabled:

- If the node implements Corrected error counters, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 1. For more information, see \( \text{ERR}<n>\text{MISC0} \).
- Otherwise, the fault handling interrupt is generated for all detected Corrected errors.

<table>
<thead>
<tr>
<th>CFI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault handling interrupt not generated for Corrected errors.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault handling interrupt generated for Corrected errors.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**When \( \text{ERR}<n>\text{FR.CFI} == 0b11: \)**

When \( \text{ERR}<n>\text{FR.CFI} == 0b11 \), this field is named RCFI.

Fault handling interrupt for Corrected errors on reads enable.

When enabled:

- If the node implements Corrected error counters, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 1. For more information, see \( \text{ERR}<n>\text{MISC0} \).
- Otherwise, the fault handling interrupt is generated for all detected Corrected errors on reads.

<table>
<thead>
<tr>
<th>RCFI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault handling interrupt not generated for Corrected errors on reads.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault handling interrupt generated for Corrected errors on reads.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.

**WUE, bit [7]**

**When \( \text{ERR}<n>\text{FR.UE} == 0b11: \)**

In-band Uncorrected error reporting on writes enable.

When enabled, responses to writes that detect an Uncorrected error that cannot be deferred are signaled in-band as a detected Uncorrected error (External abort).
<table>
<thead>
<tr>
<th>WUE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External abort response for Uncorrected errors disabled for writes.</td>
</tr>
<tr>
<td>0b1</td>
<td>External abort response for Uncorrected errors enabled for writes.</td>
</tr>
</tbody>
</table>

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**WFI, bit [6]**

*When ERR<n>FR.FI == 0b11:*

Fault handling interrupt on writes enable.

When enabled:

- The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors on writes.
- If the fault handling interrupt for Corrected errors control is not implemented:
  - If the node implements Corrected error counters, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to 1.
  - Otherwise, the fault handling interrupt is also generated for all detected Corrected errors on writes.

<table>
<thead>
<tr>
<th>WFI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault handling interrupt disabled on writes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault handling interrupt enabled on writes.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**WUI, bit [5]**

*When ERR<n>FR.UI == 0b11:*

Uncorrected error recovery interrupt on writes enable.

When enabled, the error recovery interrupt is generated for all detected Uncorrected errors on writes that are not deferred.

<table>
<thead>
<tr>
<th>WUI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error recovery interrupt disabled on writes.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error recovery interrupt enabled on writes.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
• On a Cold reset, this field resets to an architecturally *UNKNOWN* value.

**Otherwise:**

Reserved, RES0.

**UE, bit [4]**

*When ERR\(<n>\)FR.UE == 0b10:*

In-band Uncorrected error reporting enable.

When enabled, responses to transactions that detect an Uncorrected error that cannot be deferred are signaled in-band as a detected Uncorrected error (External abort).

<table>
<thead>
<tr>
<th>UE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External abort response for Uncorrected errors disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>External abort response for Uncorrected errors enabled.</td>
</tr>
</tbody>
</table>

**Note**

This control applies to errors arising from both reads and writes.

The following resets apply:

• This bit is preserved on an Error Recovery reset.

• On a Cold reset, this field resets to an architecturally *UNKNOWN* value.

*When ERR\(<n>\)FR.UE == 0b11:*

When ERR\(<n>\)FR.UE == 0b11, this field is named RUE.

In-band Uncorrected error reporting on reads enable.

When enabled, responses to reads that detect an Uncorrected error that cannot be deferred are signaled in-band as a detected Uncorrected error (External abort).

<table>
<thead>
<tr>
<th>RUE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>External abort response for Uncorrected errors disabled for reads.</td>
</tr>
<tr>
<td>0b1</td>
<td>External abort response for Uncorrected errors enabled for reads.</td>
</tr>
</tbody>
</table>

The following resets apply:

• This bit is preserved on an Error Recovery reset.

• On a Cold reset, this field resets to an architecturally *UNKNOWN* value.

**Otherwise:**

Reserved, RES0.

**FI, bit [3]**

*When ERR\(<n>\)FR.FI == 0b10:*

Fault handling interrupt enable. This control applies to errors arising from both reads and writes.

When enabled:

• The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors.

• If the fault handling interrupt for Corrected errors control is not implemented:
If the node implements Corrected error counters, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to 1.

Otherwise, the fault handling interrupt is also generated for all detected Corrected errors.

<table>
<thead>
<tr>
<th>FI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault handling interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault handling interrupt enabled.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When \( \text{ERR}_n\text{FR}.FI = 0b11 \):

When \( \text{ERR}_n\text{FR}.FI = 0b11 \), this field is named \( RFI \).

Fault handling interrupt on reads enable.

When enabled:

- The fault handling interrupt is generated for all detected Deferred errors and Uncorrected errors on reads.
- If the fault handling interrupt for Corrected errors control is not implemented:
  - If the node implements Corrected error counters, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to 1.
  - Otherwise, the fault handling interrupt is also generated for all detected Corrected errors on reads.

<table>
<thead>
<tr>
<th>RFI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Fault handling interrupt disabled on reads.</td>
</tr>
<tr>
<td>0b1</td>
<td>Fault handling interrupt enabled on reads.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, \( \text{RES0} \).

\( UI, \text{bit [2]} \)

When \( \text{ERR}_n\text{FR}.UI = 0b10 \):

Uncorrected error recovery interrupt enable. This control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all detected Uncorrected errors that are not deferred.

<table>
<thead>
<tr>
<th>UI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error recovery interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error recovery interrupt enabled.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.
When \texttt{ERR<n>FR.UI} == 0b11:

When \texttt{ERR<n>FR.UI} == 0b11, this field is named RUI.

Uncorrected error recovery interrupt on reads enable.

When enabled, the error recovery interrupt is generated for all detected Uncorrected errors on reads that are not deferred.

<table>
<thead>
<tr>
<th>RUI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error recovery interrupt disabled on reads.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error recovery interrupt enabled on reads.</td>
</tr>
</tbody>
</table>

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

**IMPLEMENTATION DEFINED, bit [1]**

IMPLEMENTATION DEFINED.

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

This bit reads as an IMPLEMENTATION DEFINED value and writes to this bit have IMPLEMENTATION DEFINED behavior.

**ED, bit [0]**

When \texttt{ERR<n>FR.ED} == 0b10:

Error reporting and logging enable.

When disabled, the node behaves as if error detection and correction are disabled, and no errors are recorded or signaled by the node. Arm recommends that, when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection.

<table>
<thead>
<tr>
<th>ED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Error reporting disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Error reporting enabled.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting is disabled. That is, even with error reporting disabled, the node might continue to silently correct errors. Uncorrectable errors might result in corrupt data being silently propagated by the node.

**Note**

If this node requires initialization after Cold reset to prevent signaling false errors, then Arm recommends this bit is set to 0 on Cold reset. This allows boot software to initialize a node without signaling errors. Software can enable error reporting after the node is initialized. If the Cold reset value is 1, the reset values of other controls in this register are IMPLEMENTATION DEFINED and should not be \texttt{UNKNOWN}.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

**Accessing the ERR\text{<n>CTRLR}**

**ERR\text{<n>CTRLR} can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x008 + 64n</td>
<td>ERR\text{&lt;n&gt;CTRLR}</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
**ERR<n>FR, Error Record Feature Register, n = 0 - 65534**

The ERR<n>FR characteristics are:

**Purpose**

Defines whether <n> is the first record owned by a node. If <n> is the first record owned by the node, also defines which of the common architecturally-defined features are implemented by the node and, of the implemented features, which are software programmable.

**Configuration**

This register is present only when RAS is implemented. Otherwise, direct accesses to ERR<n>FR are RES0.

Present only if error record <n> is implemented. Otherwise, this register is RES0.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If <n> is the first error record owned by a node, then ERR<n>FR.ED != 0b00.

If <n> is not the first error record owned by a node, then ERR<n>FR.ED == 0b00.

**Attributes**

ERR<n>FR is a 64-bit register.

**Field descriptions**

The ERR<n>FR bit assignments are:

**When ERR<n>FR.ED != 0b00:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31:26</td>
<td>Reserved for identifying IMPLEMENTATION DEFINED controls</td>
</tr>
<tr>
<td>25:24</td>
<td>TS, bits [25:24]</td>
</tr>
</tbody>
</table>

**When ARMv8.4-RAS is implemented:**

Timestamp Extension.
Indicates whether, for each error record \(<m>\) owned by this node, \(\text{ERR}<m>\text{MISC3}\) is used as the timestamp register, and, if it is, the timebase used by the timestamp.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The node does not support a timestamp register.</td>
</tr>
<tr>
<td>0b01</td>
<td>The node implements a timestamp register. The timestamp uses the same timebase as the system Generic Timer.</td>
</tr>
</tbody>
</table>

\[\text{Note}\]
For an error record which has an affinity to a PE, this is the same timer that is visible through \(\text{CNTPTC_EL0}\) at the highest Exception level on that PE.

| 0b10 | The node implements a timestamp register. The timebase for the timestamp is \text{IMPLEMENTATION DEFINED}. |

All other values are reserved.

**Otherwise:**
Reserved, \(\text{RES0}\).

**CI, bits [23:22]**
Critical error interrupt.

Indicates whether the critical error interrupt and associated controls are implemented.

<table>
<thead>
<tr>
<th>CI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Does not support critical error interrupt.</td>
</tr>
<tr>
<td>0b01</td>
<td>Critical error interrupt is supported and is always enabled.</td>
</tr>
<tr>
<td>0b10</td>
<td>Critical error interrupt is supported and it can be enabled using associated controls.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**INJ, bits [21:20]**

When ARMv8.4-RAS is implemented:

Fault Injection Extension.

Indicates whether the RAS Common Fault Injection Model Extension is implemented.

<table>
<thead>
<tr>
<th>INJ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The node does not support the RAS Common Fault Injection Model Extension.</td>
</tr>
<tr>
<td>0b01</td>
<td>The node implements the RAS Common Fault Injection Model Extension. See (\text{ERR}&lt;n&gt;\text{PFGF}) for more information.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Otherwise:**
Reserved, \(\text{RES0}\).

**CEO, bits [19:18]**

When \(\text{ERR}<n>\text{FR.CEC} != 0b00\):
Corrected Error overwrite.

Indicates the behavior when a second Corrected error is detected after a first Corrected error has been recorded by an error record \(<m>\) owned by the node.
### CEO

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00 Counts Corrected errors if a counter is implemented. Keeps the previous error syndrome. If the counter overflows, or no counter is implemented, then <code>ERR&lt; m &gt; STATUS. OF</code> is set to 1.</td>
</tr>
<tr>
<td>0b01 Counts Corrected errors. If <code>ERR&lt; m &gt; STATUS. OF</code> == 1 before the Corrected error is counted, then keeps the previous syndrome. Otherwise, the previous syndrome is overwritten. If the counter overflows, then <code>ERR&lt; m &gt; STATUS. OF</code> is set to 1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Otherwise:**

Reserved, RES0.

### DUI, bits [17:16]

**When ERR< n > FR.UI != 0b00:**

Error recovery interrupt for deferred errors.

Indicates whether the node implements a control for enabling error recovery interrupts on deferred errors.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00 Does not support feature. <code>ERR&lt; n &gt; CTRL. DUI</code> is RES0.</td>
</tr>
<tr>
<td>0b10 Feature is controllable using <code>ERR&lt; n &gt; CTRL. DUI</code>.</td>
</tr>
<tr>
<td>0b11 Feature is controllable using <code>ERR&lt; n &gt; CTRL. WDUI</code> for writes and <code>ERR&lt; n &gt; CTRL. RDUI</code> for reads.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Otherwise:**

Reserved, RES0.

### RP, bit [15]

**When ERR< n > FR.CEC != 0b00:**

Repeat counter.

Indicates whether the node implements a repeat Corrected error counter in `ERR< m > MISC0` for each error record <m> owned by the node that implements a standard Corrected error counter.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 A single CE counter is implemented.</td>
</tr>
<tr>
<td>0b1 A first (repeat) counter and a second (other) counter are implemented. The repeat counter is the same size as the primary error counter.</td>
</tr>
</tbody>
</table>

**Otherwise:**

Reserved, RES0.

### CEC, bits [14:12]

Corrected Error Counter.

Indicates whether the node implements standard Corrected error counter (CE counter) mechanisms in `ERR< m > MISC0` for each error record <m> owned by the node that can record countable errors.
CEC

<table>
<thead>
<tr>
<th>CEC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Does not implement the standard Corrected error counter model.</td>
</tr>
<tr>
<td>0b01</td>
<td>Implies an 8-bit Corrected error counter in ERR&lt;m&gt;MISC0[39:32].</td>
</tr>
<tr>
<td>0b10</td>
<td>Implements a 16-bit Corrected error counter in ERR&lt;m&gt;MISC0[47:32].</td>
</tr>
</tbody>
</table>

All other values are reserved.

**Note**

Implementations might include other error counter models, or might include the standard model and not indicate this in ERR<n>FR.

**CFI, bits [11:10]**

When ERR<n>FR.FI != 0b00:

Fault handling interrupt for corrected errors.

Indicates whether the node implements a control for enabling fault handling interrupts on corrected errors.

<table>
<thead>
<tr>
<th>CFI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Does not support feature. ERR&lt;n&gt;CTLR.CFI is RES0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Feature is controllable using ERR&lt;n&gt;CTLR.CFI.</td>
</tr>
<tr>
<td>0b11</td>
<td>Feature is controllable using ERR&lt;n&gt;CTLR.WCFI for writes and ERR&lt;n&gt;CTLR.RCFI for reads.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**UE, bits [9:8]**

In-band uncorrected error reporting.

Indicates whether the node implements in-band uncorrected error reporting (External aborts), and, if so, whether the node implements controls for enabling and disabling the reporting.

<table>
<thead>
<tr>
<th>UE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Does not support feature. ERR&lt;n&gt;CTLR.UE is RES0.</td>
</tr>
<tr>
<td>0b01</td>
<td>Feature always enabled. ERR&lt;n&gt;CTLR.UE is RES0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Feature is controllable using ERR&lt;n&gt;CTLR.UE.</td>
</tr>
<tr>
<td>0b11</td>
<td>Feature is controllable using ERR&lt;n&gt;CTLR.WUE for writes and ERR&lt;n&gt;CTLR.RUE for reads.</td>
</tr>
</tbody>
</table>

**FI, bits [7:6]**

Fault handling interrupt.

Indicates whether the node implements a fault handling interrupt, and, if so, whether the node implements controls for enabling and disabling the interrupt.

<table>
<thead>
<tr>
<th>FI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Does not support feature. ERR&lt;n&gt;CTLR.FI is RES0.</td>
</tr>
<tr>
<td>0b01</td>
<td>Feature always enabled. ERR&lt;n&gt;CTLR.FI is RES0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Feature is controllable using ERR&lt;n&gt;CTLR.FI.</td>
</tr>
<tr>
<td>0b11</td>
<td>Feature is controllable using ERR&lt;n&gt;CTLR.WFI for writes and ERR&lt;n&gt;CTLR.RFI for reads.</td>
</tr>
</tbody>
</table>
UI, bits [5:4]

Error recovery interrupt for uncorrected errors.

Indicates whether the node implements an error recovery interrupt, and, if so, whether the node implements controls for enabling and disabling the interrupt.

<table>
<thead>
<tr>
<th>UI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Does not support feature. ERR&lt;n&gt;CTRL.UI is RES0.</td>
</tr>
<tr>
<td>0b01</td>
<td>Feature always enabled. ERR&lt;n&gt;CTRL.UI is RES0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Feature is controllable using ERR&lt;n&gt;CTRL.UI.</td>
</tr>
<tr>
<td>0b11</td>
<td>Feature is controllable using ERR&lt;n&gt;CTRL.WUI for writes and ERR&lt;n&gt;CTRL.RUI for reads.</td>
</tr>
</tbody>
</table>

IMPLEMENTATION DEFINED, bits [3:2]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value.

ED, bits [1:0]

Error reporting and logging.

Indicates whether <n> is the first record owned by the node, and, if it is, whether the node implements controls for enabling and disabling error reporting and logging.

<table>
<thead>
<tr>
<th>ED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b01</td>
<td>Feature always enabled. ERR&lt;n&gt;CTRL.ED is RES0.</td>
</tr>
<tr>
<td>0b10</td>
<td>Feature is controllable using ERR&lt;n&gt;CTRL.ED.</td>
</tr>
</tbody>
</table>

The value 0b11 is reserved.

When ERR<n>FR.ED == 0b00:

<table>
<thead>
<tr>
<th>Bits [63:2]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
</table>

ED, bits [1:0]

Error reporting and logging.

Indicates whether <n> is the first record owned by the node, and, if it is, whether the node implements controls for enabling and disabling error reporting and logging.

<table>
<thead>
<tr>
<th>ED</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Error record &lt;n&gt; is not the first record owned by the node.</td>
</tr>
</tbody>
</table>

Accessing the ERR<n>FR

ERR<n>FR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x000 + 64n</td>
<td>ERR&lt;n&gt;FR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERR<n>FR, Error Record Feature Register, n = 0 - 65534
ERR<n>MISC0, Error Record Miscellaneous Register 0

The ERR<n>MISC0 characteristics are:

**Purpose**

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- A Corrected error counter or counters.
- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- Other state information not present in the corresponding status and address registers.

If the node <q> that owns error record <n> implements architecturally-defined error counters, so that ERR<q>FR. CEC != 0b000, and error record <n> can record countable errors, then ERR<n>MISC0 implements the architecturally-defined error counter or counters.

**Configuration**

Some or all RW fields of this register have defined reset values.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If error record <n> is not implemented, ERR<n>MISC0 is RES0.

ERR<q>FR describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

---

**Note**

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.

---

**Attributes**

ERR<n>MISC0 is a 64-bit register.

**Field descriptions**

The ERR<n>MISC0 bit assignments are:
When ERR<q>FR.CEC == 0b000:

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

When ERR<q>FR.CEC == 0b100 and ERR<q>FR.RP == 0:

IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

OF, bit [47]

Sticky overflow bit.

Set to 1 when the Corrected error count field is incremented and wraps through zero.

<table>
<thead>
<tr>
<th>OF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Counter has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Counter has overflowed.</td>
</tr>
</tbody>
</table>

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS. OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [46:32]

Corrected error count.

Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.
IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**When ERR<q>FR.CEC == 0b010 and ERR<q>FR.RP == 0:**

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| IMPLEMENTATION DEFINED | OF | CEC |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

**IMPLEMENTATION DEFINED, bits [63:40]**

**OF, bit [39]**

Sticky overflow bit.

Set to 1 when the Corrected error count field is incremented and wraps through zero.

<table>
<thead>
<tr>
<th>OF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Counter has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Counter has overflowed.</td>
</tr>
</tbody>
</table>

A direct write that modifies this bit might indirectly set ERR<n>STATUS. OF to an UNKNOWN value and a direct write to ERR<n>STATUS. OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**CEC, bits [38:32]**

Corrected error count.

Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**IMPLEMENTATION DEFINED, bits [31:0]**

**IMPLEMENTATION DEFINED.**

**IMPLEMENTATION DEFINED syndrome.** This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**When ERR<q>FR.CEC == 0b100 and ERR<q>FR.RP == 1:**

| 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| OF | CEC | DFR | CECR |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

Page 3526
OFO, bit [63]

Sticky overflow bit, other.

Set to 1 when the Corrected error count, other, field is incremented and wraps through zero.

<table>
<thead>
<tr>
<th>OFO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Other counter has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Other counter has overflowed.</td>
</tr>
</tbody>
</table>

A direct write that modifies this bit might indirectly set \texttt{ERR<n>STATUS.OF} to an \texttt{UNKNOWN} value and a direct write to \texttt{ERR<n>STATUS.OF} that clears it to zero might indirectly set this bit to an \texttt{UNKNOWN} value.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

CECO, bits [62:48]

Corrected error count, other.

Incremented for each countable error that is not accounted for by incrementing \texttt{ERR<n>MISC0.CECR}.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

OFR, bit [47]

Sticky overflow bit, repeat.

Set to 1 when the Corrected error count, repeat, field is incremented and wraps through zero.

<table>
<thead>
<tr>
<th>OFR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Repeat counter has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Repeat counter has overflowed.</td>
</tr>
</tbody>
</table>

A direct write that modifies this bit might indirectly set \texttt{ERR<n>STATUS.OF} to an \texttt{UNKNOWN} value and a direct write to \texttt{ERR<n>STATUS.OF} that clears it to zero might indirectly set this bit to an \texttt{UNKNOWN} value.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally \texttt{UNKNOWN} value.

CECR, bits [46:32]

Corrected error count, repeat.

Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is \texttt{IMPLEMENTATION DEFINED} and might be \texttt{UNPREDICTABLE} whether Deferred and Uncorrected errors are countable errors.

Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the \texttt{IMPLEMENTATION DEFINED} \texttt{ERR<n>MISC<n>} fields on a first Corrected error. \texttt{ERR<n>MISC0.CECR} is then incremented for each subsequent Corrected Error in the same set and way.
The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**IMPLEMENTATION DEFINED, bits [31:0]**

**IMPLEMENTATION DEFINED.**

**IMPLEMENTATION DEFINED** syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**When ERR<q>FR.CEC == 0b010 and ERR<q>FR.RP == 1:**

<table>
<thead>
<tr>
<th></th>
<th>IMPLEMENTATION DEFINED</th>
<th>OFO</th>
<th>CECO</th>
<th>OFR</th>
<th>CECR</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [63:48]**

**IMPLEMENTATION DEFINED.**

**IMPLEMENTATION DEFINED** syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**OFO, bit [47]**

Sticky overflow bit, other.

Set to 1 when the Corrected error count, other, field is incremented and wraps through zero.

<table>
<thead>
<tr>
<th>OFO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Other counter has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Other counter has overflowed.</td>
</tr>
</tbody>
</table>

A direct write that modifies this bit might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this bit to an UNKNOWN value.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**CECO, bits [46:40]**

Corrected error count, other.

Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**OFR, bit [39]**

Sticky overflow bit, repeat.

Set to 1 when the Corrected error count, repeat, field is incremented and wraps through zero.
### OFR

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00                       Repeat counter has not overflowed.</td>
</tr>
<tr>
<td>01                       Repeat counter has overflowed.</td>
</tr>
</tbody>
</table>

A direct write that modifies this bit might indirectly set \( \text{ERR}<n>\text{STATUS}.\text{OF} \) to an \textit{UNKNOWN} value and a direct write to \( \text{ERR}<n>\text{STATUS}.\text{OF} \) that clears it to zero might indirectly set this bit to an \textit{UNKNOWN} value.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally \textit{UNKNOWN} value.

### CECR, bits [38:32]

Corrected error count, repeat.

Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is \textit{IMPLEMENTATION DEFINED} and might be \textit{UNPREDICTABLE} whether Deferred and Uncorrected errors are countable errors.

#### Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the \textit{IMPLEMENTATION DEFINED} \( \text{ERR}<n>\text{MISC}<m> \) fields on a first Corrected error. \( \text{ERR}<n>\text{MISC}0.\text{CECR} \) is then incremented for each subsequent Corrected Error in the same set and way.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally \textit{UNKNOWN} value.

### IMPLEMENTATION DEFINED, bits [31:0]

\textit{IMPLEMENTATION DEFINED}.

\textit{IMPLEMENTATION DEFINED} syndrome. This field reads as an \textit{IMPLEMENTATION DEFINED} value and writes to this field have \textit{IMPLEMENTATION DEFINED} behavior.

### Accessing the \( \text{ERR}<n>\text{MISC}0 \)

Arm recommends that a miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write. When \( \text{ERR}<n>\text{STATUS}.\text{MV} \) is set to 1, the miscellaneous syndrome for the most recently recorded error should ignore writes.

\( \text{ERR}<n>\text{MISC}0 \) can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x020 + 64n</td>
<td>( \text{ERR}&lt;n&gt;\text{MISC}0 )</td>
</tr>
</tbody>
</table>

Accesses on this interface are \textit{RW}.
ERR<n>MISC1, Error Record Miscellaneous Register 1, n = 0 - 65534

The ERR<n>MISC1 characteristics are:

**Purpose**

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers can contain:

- A Corrected error counter or counters.
- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- Other state information not present in the corresponding status and address registers.

**Configuration**

This register is present only when RAS is implemented. Otherwise, direct accesses to ERR<n>MISC1 are UNDEFINED.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If error record <n> is not implemented, ERR<n>MISC1 is RES0.

ERR<q>FR describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

---

**Note**

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.

---

**Attributes**

ERR<n>MISC1 is a 64-bit register.

**Field descriptions**

The ERR<n>MISC1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>32</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>
IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Accessing the ERR\textsubscript{<n>}-MISC1

Arm recommends that a miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.

When ERR\textsubscript{<n>}-STATUS.MV is set to 1, the miscellaneous syndrome for the most recently recorded error should ignore writes.

ERR\textsubscript{<n>}-MISC1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x028 + 64n</td>
<td>ERR\textsubscript{&lt;n&gt;}-MISC1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERR<n>MISC2, Error Record Miscellaneous Register 2, 
\( n = 0 - 65534 \)

The ERR<n>MISC2 characteristics are:

**Purpose**

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers can contain:

- A Corrected error counter or counters.
- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- Other state information not present in the corresponding status and address registers.

**Configuration**

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERR<n>MISC2 are RES0.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If error record \( <n> \) is not implemented, ERR<n>MISC2 is RES0.

ERR<q>FR describes the features implemented by the node that owns error record \( <n> \). \( <q> \) is the index of the first error record owned by the same node as error record \( <n> \). If the node owns a single record, then \( q = n \).

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

**Note**

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where \( q \) is the index of the first error record owned by the same node as error record \( n \). If the node owns a single record then \( q = n \).

**Attributes**

ERR<n>MISC2 is a 64-bit register.

**Field descriptions**

The ERR<n>MISC2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>62</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>29</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>28</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>27</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>26</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>25</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>24</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>23</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>22</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>21</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>20</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>19</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>18</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>17</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>16</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>15</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>14</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>13</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>12</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>11</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>10</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>9</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>8</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>7</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>6</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>5</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>4</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>3</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>2</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>1</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>
IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**Accessing the ERR\(<n>\)MISC2**

Arm recommends that a miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.

When ERR\(<n>\)STATUS.MV is set to 1, the miscellaneous syndrome for the most recently recorded error should ignore writes.

**ERR\(<n>\)MISC2 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFFFFFFFF + 64n</td>
<td>ERR(&lt;n&gt;)MISC2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERR<n>MISC3, Error Record Miscellaneous Register 3, n = 0 - 65534

The ERR<n>MISC3 characteristics are:

**Purpose**

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers can contain:

- A Corrected error counter or counters.
- Information to identify the FRU in which the error was detected, and might contain enough information to locate the error within that FRU.
- Other state information not present in the corresponding status and address registers.

If the node <q> that owns error record <n> supports the RAS Timestamp Extension, then ERR<n>MISC3 contains the timestamp value for error record <n> when the error was detected. Otherwise the contents of ERR<n>MISC3 are IMPLEMENTATION DEFINED.

**Configuration**

External register ERR<n>MISC3 is architecturally mapped to AArch64 System register ERXMISC3_EL1 when ERRSELR_EL1.SEL == n.

External register ERR<n>MISC3 bits [31:0] are architecturally mapped to AArch32 System register ERXMISC6[31:0] when ERRSELR_EL1.SEL == n.

External register ERR<n>MISC3 bits [63:32] are architecturally mapped to AArch32 System register ERXMISC7[31:0] when ERRSELR_EL1.SEL == n.

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERR<n>MISC3 are RES0.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If error record <n> is not implemented, ERR<n>MISC3 is RES0.

ERR<q>FR describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

---

**Note**

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.
**Attributes**

ERR<n>MISC3 is a 64-bit register.

**Field descriptions**

The ERR<n>MISC3 bit assignments are:

**When ERR<q>FR.TS != 0b00:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| TS | TS |

**TS, bits [63:0]**

Timestamp.

Timestamp value recorded when the error was detected. Valid only if ERR<n>STATUS.V == 1.

See ERR<n>FR.TS.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

Access to this field is RO or RW.

**When ERR<q>FR.TS == 0b00:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| IMPLEMENTATION DEFINED | IMPLEMENTATION DEFINED |

**IMPLEMENTATION DEFINED, bits [63:0]**

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED syndrome. This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**Accessing the ERR<n>MISC3**

Arm recommends that a miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.

When ERR<n>STATUS.MV is set to 1, the miscellaneous syndrome for the most recently recorded error should ignore writes.

ERR<n>MISC3 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x038 + 64n</td>
<td>ERR&lt;n&gt;MISC3</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
The ERR<n>PFGCDN characteristics are:

**Purpose**

Generates one of the errors enabled in the corresponding ERR<n>PFGCTL register.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERR<n>PFGCDN are RES0.

Present only when the RAS Common Fault Injection Model Extension is implemented by this node so that ERR<n>FR.INJ != 0b00, error record <n> is implemented, and error record <n> is the first error record owned by a node. Otherwise, RES0.

ERR<n>FR describes the features implemented by the node.

**Attributes**

ERR<n>PFGCDN is a 64-bit register.

**Field descriptions**

The ERR<n>PFGCDN bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    | RES0|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |  9 |  8 |  7 |  6 |  5 |  4 |  3 |  2 |  1 |  0|

**Bits [63:32]**

Reserved, RES0.

**CDN, bits [31:0]**

Countdown value.

This field is copied to Error Generation Counter when either:

- Software writes ERR<n>PFGCTL.CDNEN with 1.
- The Error Generation Counter decrements to zero and ERR<n>PFGCTL.R == 1.

While ERR<n>PFGCTL.CDNEN == 1 and the Error Generation Counter is nonzero, the counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate. When the counter reaches 0, one of the errors enabled in the ERR<n>PFGCTL register is generated.

**Note**

The current Error Generation Counter value is not visible to software.

On a Cold reset, this field resets to an architecturally UNKNOWN value.
Accessing the **ERR<n>PFGCDN**

**ERR<n>PFGCDN can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x810 + 64n</td>
<td>ERR&lt;n&gt;PFGCDN</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
ERR<n>PFGCTL, Pseudo-fault Generation Control Register, n = 0 - 65534

The ERR<n>PFGCTL characteristics are:

**Purpose**

Enables controlled fault generation.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERR<n>PFGCTL are RES0.

Present only when the RAS Common Fault Injection Model Extension is implemented by this node so that ERR<n>FR.INJ != 0b00, error record <n> is implemented, and error record <n> is the first error record owned by a node. Otherwise, RES0.

ERR<n>FR describes the features implemented by the node.

**Attributes**

ERR<n>PFGCTL is a 64-bit register.

**Field descriptions**

The ERR<n>PFGCTL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>CDNEN</td>
<td>R</td>
</tr>
<tr>
<td>61</td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**CDNEN, bit [31]**

Countdown Enable. Controls transfers from the value that is held in the ERR<n>PFGCDN into the Error Generation Counter, and enables this counter.

<table>
<thead>
<tr>
<th>CDNEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Error Generation Counter is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Error Generation Counter is enabled. On a write of 1 to this bit, the Error Generation Counter is set to ERR&lt;n&gt;PFGCDN.CDN.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to 0.

**R, bit [30]**

Restart. Controls whether, on reaching zero, the Error Generation Counter restarts from the ERR<n>PFGCDN value, or stops.
Meaning

0b0 On reaching 0, the Error Generation Counter stops.
0b1 On reaching 0, the Error Generation Counter is set to ERR<n>PFGCDN.CDN.

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [29:13]**

Reserved, RES0.

**MV, bit [12]**

Miscellaneous syndrome. The value that is written to ERR<n>STATUS.MV when an injected error is recorded.

<table>
<thead>
<tr>
<th>MV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS.MV is set to 0 when an injected error is recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS.MV is set to 1 when an injected error is recorded.</td>
</tr>
</tbody>
</table>

This bit reads-as-one if the node always records some syndrome in ERR<n>MISC<m>, setting ERR<n>STATUS.MV to 1, when an injected error is recorded.

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**AV, bit [11]**

Address syndrome. The value that is written to ERR<n>STATUS.AV when an injected error is recorded.

<table>
<thead>
<tr>
<th>AV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS.AV is set to 0 when an injected error is recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS.AV is set to 1 when an injected error is recorded.</td>
</tr>
</tbody>
</table>

This bit reads-as-one if the node always sets ERR<n>STATUS.AV to 1 when an injected error is recorded.

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**PN, bit [10]**

Poison flag. The value that is written to ERR<n>STATUS.PN when an injected error is recorded.

<table>
<thead>
<tr>
<th>PN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS.PN is set to 0 when an injected error is recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS.PN is set to 1 when an injected error is recorded.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**ER, bit [9]**

Error Reported flag. The value that is written to ERR<n>STATUS.ER when an injected error is recorded.

<table>
<thead>
<tr>
<th>ER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS.ER is set to 0 when an injected error is recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS.ER is set to 1 when an injected error is recorded.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the node does not support this control.
On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**CI, bit [8]**

Critical Error flag. The value that is written to **ERR<n>STATUS.CI** when an injected error is recorded.

<table>
<thead>
<tr>
<th>CI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>ERR&lt;n&gt;STATUS.CI</strong> is set to 0 when an injected error is recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>ERR&lt;n&gt;STATUS.CI</strong> is set to 1 when an injected error is recorded.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if the node does not support this control.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**CE, bits [7:6]**

Corrected Error generation enable. Controls the type of Corrected Error condition that might be generated.

<table>
<thead>
<tr>
<th>CE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>No error of this type is generated.</td>
</tr>
<tr>
<td>0b01</td>
<td>A non-specific Corrected Error, that is, a Corrected Error that is</td>
</tr>
<tr>
<td></td>
<td>recorded as <strong>ERR&lt;n&gt;STATUS.CE</strong> == 0b10, might be generated</td>
</tr>
<tr>
<td></td>
<td>when the Error Generation Counter decrements to zero.</td>
</tr>
<tr>
<td>0b10</td>
<td>A transient Corrected Error, that is, a Corrected Error that is</td>
</tr>
<tr>
<td></td>
<td>recorded as <strong>ERR&lt;n&gt;STATUS.CE</strong> == 0b01, might be generated</td>
</tr>
<tr>
<td></td>
<td>when the Error Generation Counter decrements to zero.</td>
</tr>
<tr>
<td>0b11</td>
<td>A persistent Corrected Error, that is, a Corrected Error that is</td>
</tr>
<tr>
<td></td>
<td>recorded as <strong>ERR&lt;n&gt;STATUS.CE</strong> == 0b11, might be generated</td>
</tr>
<tr>
<td></td>
<td>when the Error Generation Counter decrements to zero.</td>
</tr>
</tbody>
</table>

The set of permitted values for this field is defined by **ERR<n>PFGF.CE**.

This field is **RES0** if the node does not support this control.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**DE, bit [5]**

Deferred Error generation enable. Controls whether this type of error condition might be generated. It is **IMPLEMENTATION DEFINED** whether the error is generated if the data is not consumed.

<table>
<thead>
<tr>
<th>DE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No error of this type is generated.</td>
</tr>
<tr>
<td>0b1</td>
<td>An error of this type might be generated when the Error</td>
</tr>
<tr>
<td></td>
<td>Generation Counter decrements to zero.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if the node does not support this control.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**UEO, bit [4]**

Latent or Restartable Error generation enable. Controls whether this type of error condition might be generated. It is **IMPLEMENTATION DEFINED** whether the error is generated if the data is not consumed.

<table>
<thead>
<tr>
<th>UEO</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No error of this type is generated.</td>
</tr>
<tr>
<td>0b1</td>
<td>An error of this type might be generated when the Error</td>
</tr>
<tr>
<td></td>
<td>Generation Counter decrements to zero.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if the node does not support this control.

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.
UER, bit [3]

Signaled or Recoverable Error generation enable. Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed.

<table>
<thead>
<tr>
<th>UER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No error of this type is generated.</td>
</tr>
<tr>
<td>0b1</td>
<td>An error of this type might be generated when the Error Generation Counter decrements to zero.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

UEU, bit [2]

Unrecoverable Error generation enable. Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed.

<table>
<thead>
<tr>
<th>UEU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No error of this type is generated.</td>
</tr>
<tr>
<td>0b1</td>
<td>An error of this type might be generated when the Error Generation Counter decrements to zero.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

UC, bit [1]

Uncontainable Error generation enable. Controls whether this type of error condition might be generated. It is IMPLEMENTATION DEFINED whether the error is generated if the data is not consumed.

<table>
<thead>
<tr>
<th>UC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No error of this type is generated.</td>
</tr>
<tr>
<td>0b1</td>
<td>An error of this type might be generated when the Error Generation Counter decrements to zero.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

OF, bit [0]

Overflow flag. The value that is written to ERR<n>STATUS.OF when an injected error is recorded.

<table>
<thead>
<tr>
<th>OF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS.OF is set to 0 when an injected error is recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS.OF is set to 1 when an injected error is recorded.</td>
</tr>
</tbody>
</table>

This bit is RES0 if the node does not support this control.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the ERR<n>PFGCTL**

ERR<n>PFGCTL can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x808 + 64n</td>
<td>ERR&lt;n&gt;PFGCTL</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
ERR\(n\) PFGCTL, Pseudo-fault Generation Control Register, \(n = 0 - 65534\)
ERR<n>PFGF, Pseudo-fault Generation Feature Register, n = 0 - 65534

The ERR<n>PFGF characteristics are:

**Purpose**

Defines which common architecturally-defined fault generation features are implemented.

**Configuration**

This register is present only when ARMv8.4-RAS is implemented. Otherwise, direct accesses to ERR<n>PFGF are RES0.

Present only when the RAS Common Fault Injection Model Extension is implemented by this node so that ERR<n>FR.INJ != 0b00, error record <n> is implemented, and error record <n> is the first error record owned by a node. Otherwise, RES0.

ERR<n>FR describes the features implemented by the node.

**Attributes**

ERR<n>PFGF is a 64-bit register.

**Field descriptions**

The ERR<n>PFGF bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 - 31</td>
<td>Reserved, RES0.</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>R</td>
<td>Restartable. Support for Error Generation Counter restart mode.</td>
</tr>
<tr>
<td>29</td>
<td>SYN</td>
<td>Syndrome. Fault syndrome injection.</td>
</tr>
<tr>
<td>31 - 0</td>
<td>MVAVPNERCIUEDEOUUEDOUUF</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:31]**

Reserved, RES0.

**R, bit [30]**

Restartable. Support for Error Generation Counter restart mode.

<table>
<thead>
<tr>
<th>R</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The node does not support this feature.</td>
</tr>
<tr>
<td>0b1</td>
<td>Feature controllable.</td>
</tr>
</tbody>
</table>

**SYN, bit [29]**

Syndrome. Fault syndrome injection.

<table>
<thead>
<tr>
<th>SYN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, the node sets</td>
</tr>
<tr>
<td></td>
<td>ERR&lt;n&gt;STATUS. {IERR, SERR} to IMPLEMENTATIONDEFINED values. ERR&lt;n&gt;STATUS. {IERR, SERR} are UNKNOWN when</td>
</tr>
<tr>
<td></td>
<td>ERR&lt;n&gt;STATUS.V == 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>When an injected error is recorded, the node does not update the</td>
</tr>
<tr>
<td></td>
<td>ERR&lt;n&gt;STATUS. {IERR, SERR} fields. ERR&lt;n&gt;STATUS. {IERR, SERR} are writable when ERR&lt;n&gt;STATUS.V == 0.</td>
</tr>
</tbody>
</table>
Note

If this bit is 1, software can write specific values into the \( \text{ERR}<n>\text{STATUS} \) \{IERR, SERR\} fields when setting up a fault injection event. The sets of values that can be written to these fields is \text{IMPLEMENTATION DEFINED}.

Bits [28:13]

Reserved, RES0.

MV, bit [12]

Miscellaneous syndrome.

Additional syndrome injection. Defines whether software can control all or part of the syndrome recorded in the \( \text{ERR}<n>\text{MISC}<m> \) registers when an injected error is recorded.

It is \text{IMPLEMENTATION DEFINED} which syndrome fields in \( \text{ERR}<n>\text{MISC}<m> \) this refers to, as some fields might always be recorded by an error. For example, a Corrected Error counter.

<table>
<thead>
<tr>
<th>MV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, the node might record \text{IMPLEMENTATION DEFINED} additional syndrome in ( \text{ERR}&lt;n&gt;\text{MISC}&lt;m&gt; ). If any syndrome is recorded in ( \text{ERR}&lt;n&gt;\text{MISC}&lt;m&gt; ), ( \text{ERR}&lt;n&gt;\text{STATUS}.\text{MV} ) is set to 1.</td>
</tr>
</tbody>
</table>
| 0b1 | When an injected error is recorded, the node does not update all the syndrome fields in the \( \text{ERR}<n>\text{MISC}<m> \) and does one of:  
  • The node does not update any fields in \( \text{ERR}<n>\text{MISC}<m> \) and sets \( \text{ERR}<n>\text{STATUS}.\text{MV} \) to \( \text{ERR}<n>\text{PGFCTL}.\text{MV} \).  
  • The node records some syndrome in \( \text{ERR}<n>\text{MISC}<m> \) and sets \( \text{ERR}<n>\text{STATUS}.\text{MV} \) to 1. \( \text{ERR}<n>\text{PGFCTL}.\text{MV} \) is \text{RAO}.  
The syndrome fields that the node does not update are unchanged and must be writable when \( \text{ERR}<n>\text{STATUS}.\text{MV} \) is set to 0. |

Note

If this bit is 1, software can write specific values into the \( \text{ERR}<n>\text{MISC}<m> \) registers when setting up a fault injection event. The values that can be written to these registers are \text{IMPLEMENTATION DEFINED}.

AV, bit [11]

Address syndrome. Address syndrome injection.

<table>
<thead>
<tr>
<th>AV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, the node either sets ( \text{ERR}&lt;n&gt;\text{ADDR} ) and ( \text{ERR}&lt;n&gt;\text{STATUS}.\text{AV} ) for the access, or leaves these unchanged.</td>
</tr>
</tbody>
</table>
| 0b1 | When an injected error is recorded, the node does not update \( \text{ERR}<n>\text{ADDR} \) and does one of:  
  • Sets \( \text{ERR}<n>\text{STATUS}.\text{AV} \) to \( \text{ERR}<n>\text{PGFCTL}.\text{AV} \).  
  • Sets \( \text{ERR}<n>\text{STATUS}.\text{AV} \) to 1. \( \text{ERR}<n>\text{PGFCTL}.\text{AV} \) is \text{RAO}. \( \text{ERR}<n>\text{ADDR} \) must be writable when \( \text{ERR}<n>\text{STATUS}.\text{AV} \) is set to 0. |

Note

If this bit is 1, software can write a specific value into \( \text{ERR}<n>\text{ADDR} \) when setting up a fault injection event.
PN, bit [10]

Poison flag. Describes how the fault generation feature of the node sets the $\text{ERR}_n\text{STATUS}.\text{PN}$ status flag.

<table>
<thead>
<tr>
<th>PN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets $\text{ERR}_n\text{STATUS}.\text{PN}$ to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>When an injected error is recorded, $\text{ERR}_n\text{STATUS}.\text{PN}$ is set to $\text{ERR}_n\text{PFGCTL}.\text{PN}$.</td>
</tr>
</tbody>
</table>

This behavior replaces the architecture-defined rules for setting the PN bit.

This bit reads-as-zero if the node does not support this flag.

ER, bit [9]

Error Reported flag. Describes how the fault generation feature of the node sets the $\text{ERR}_n\text{STATUS}.\text{ER}$ status flag.

<table>
<thead>
<tr>
<th>ER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, the node sets $\text{ERR}_n\text{STATUS}.\text{ER}$ according to the architecture-defined rules for setting the ER bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>When an injected error is recorded, $\text{ERR}_n\text{STATUS}.\text{ER}$ is set to $\text{ERR}_n\text{PFGCTL}.\text{ER}$. This behavior replaces the architecture-defined rules for setting the ER bit.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this flag.

CI, bit [8]

Critical Error flag. Describes how the fault generation feature of the node sets the $\text{ERR}_n\text{STATUS}.\text{CI}$ status flag.

<table>
<thead>
<tr>
<th>CI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets $\text{ERR}_n\text{STATUS}.\text{CI}$ to 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>When an injected error is recorded, $\text{ERR}_n\text{STATUS}.\text{CI}$ is set to $\text{ERR}_n\text{PFGCTL}.\text{CI}$.</td>
</tr>
</tbody>
</table>

This behavior replaces the architecture-defined rules for setting the CI bit.

This bit reads-as-zero if the node does not support this flag.

CE, bits [7:6]

Corrected Error generation. Describes the types of Corrected Error that the fault generation feature of the node can generate.

<table>
<thead>
<tr>
<th>CE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The fault generation feature of the node cannot generate this type of error.</td>
</tr>
<tr>
<td>0b01</td>
<td>The fault generation feature of the node allows generation of a non-specific Corrected Error, that is, a Corrected Error that is recorded as $\text{ERR}_n\text{STATUS}.\text{CE} == 0b10$.</td>
</tr>
<tr>
<td>0b11</td>
<td>The fault generation feature of the node allows generation of transient or persistent Corrected Errors, that is, Corrected Errors that are recorded as $\text{ERR}_n\text{STATUS}.\text{CE} == 0b01$ and $0b11$.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This bit reads-as-zero if the node does not support this type of error.

DE, bit [5]

Deferred Error generation. Describes whether the fault generation feature of the node can generate this type of error.
### DE, bit [0]

Meaning

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault generation feature of the node cannot generate this type of error.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault generation feature of the node allows generation of this type of error.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this type of error.

### UEO, bit [4]

Latent or Restartable Error generation. Describes whether the fault generation feature of the node can generate this type of error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault generation feature of the node cannot generate this type of error.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault generation feature of the node allows generation of this type of error.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this type of error.

### UER, bit [3]

Signaled or Recoverable Error generation. Describes whether the fault generation feature of the node can generate this type of error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault generation feature of the node cannot generate this type of error.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault generation feature of the node allows generation of this type of error.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this type of error.

### UEU, bit [2]

Unrecoverable Error generation. Describes whether the fault generation feature of the node can generate this type of error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault generation feature of the node cannot generate this type of error.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault generation feature of the node allows generation of this type of error.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this type of error.

### UC, bit [1]

Uncontainable Error generation. Describes whether the fault generation feature of the node can generate this type of error.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The fault generation feature of the node cannot generate this type of error.</td>
</tr>
<tr>
<td>0b1</td>
<td>The fault generation feature of the node allows generation of this type of error.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this type of error.

### OF, bit [0]

Overflow flag. Describes how the fault generation feature of the node sets the \texttt{ERR<n>STATUS} OF status flag.
<table>
<thead>
<tr>
<th>OF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When an injected error is recorded, the node sets <em>ERR&lt;n&gt;STATUS</em>.OF according to the architecture-defined rules for setting the OF bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>When an injected error is recorded, <em>ERR&lt;n&gt;STATUS</em>.OF is set to <em>ERR&lt;n&gt;PFGCTL</em>.OF. This behavior replaces the architecture-defined rules for setting the OF bit.</td>
</tr>
</tbody>
</table>

This bit reads-as-zero if the node does not support this flag.

### Accessing the ERR<n>PFGF

**ERR<n>PFGF can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x800 + 64n</td>
<td>ERR&lt;n&gt;PFGF</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The ERR<n>STATUS characteristics are:

**Purpose**

Contains status information for the error record, including:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a master.
- Whether any detected error was not corrected and deferred.
- Whether an error record has been discarded because additional errors have been detected before the first error was handled by software (overflow).
- Whether any error has been reported.
- Whether the other error record registers contain valid information.
- Whether the error was recorded because poison data was detected or because a corrupt value was detected by an error detection code.
- A primary error code.
- An IMPLEMENTATION DEFINED extended error code.

Within this register:

- The \{AV, V, MV\} bits are valid bits that define whether the error record registers are valid.
- The \{UE, OF, CE, DE, UET\} bits encode the type of error or errors recorded.
- The \{CI, ER, PN, IERR, SERR\} fields are syndrome fields.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERR<n>STATUS are UNDEFINED.

The number of error records that are implemented is IMPLEMENTATION DEFINED.

If error record <n> is not implemented, ERR<n>STATUS is RES0.

ERR<q>FR describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in this register, writing zero must always be supported to return the error record to an initial state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero and ignore writes are compliant with this requirement.

**Note**

Arm recommends that any IMPLEMENTATION DEFINED syndrome fields that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request are disabled at Cold reset and are enabled by
software writing an IMPLEMENTATION DEFINED non-zero value to an IMPLEMENTATION DEFINED field in ERR<q>CTRL, where q is the index of the first error record owned by the same node as error record n. If the node owns a single record then q = n.

**Attributes**

ERR<n>STATUS is a 64-bit register.

**Field descriptions**

The ERR<n>STATUS bit assignments are:

**When ARMv8.4-RAS is implemented:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>Address Valid</td>
</tr>
<tr>
<td>30</td>
<td>Status Register Valid</td>
</tr>
<tr>
<td>29</td>
<td>Uncorrected error</td>
</tr>
<tr>
<td>28</td>
<td>Error Record Primary Status Register, n = 0 - 65534</td>
</tr>
</tbody>
</table>

**AV, bit [31]**

Address Valid.

<table>
<thead>
<tr>
<th>AV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;ADDR not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;ADDR contains an address associated with the highest priority error recorded by this record.</td>
</tr>
</tbody>
</table>

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to 0.

**V, bit [30]**

Status Register Valid.

<table>
<thead>
<tr>
<th>V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS valid. At least one error has been recorded.</td>
</tr>
</tbody>
</table>

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to 0.

**UE, bit [29]**

Uncorrected error.
### ER, bit [28]

**Error Reported.**

<table>
<thead>
<tr>
<th>ER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No in-band error (External abort) reported.</td>
</tr>
<tr>
<td>0b1</td>
<td>An External abort was signaled by the node to the master making the access or other transaction. This can be because any of the following are true:</td>
</tr>
<tr>
<td></td>
<td>* The applicable one of the $ERR&lt;q&gt;$CTRLR.{WUE,RUE,UE} bits is implemented, and was set to 1 when an Uncorrected error was detected.</td>
</tr>
<tr>
<td></td>
<td>* The applicable one of the $ERR&lt;q&gt;$CTRLR.{WUE,RUE,UE} bits is not implemented.</td>
</tr>
</tbody>
</table>

It is **IMPLEMENTATION DEFINED** whether this bit can be set to 1 by a Deferred error.

When clearing \$ERR<n>\$STATUS.V to 0, if this bit is nonzero, then software must write one to this bit to clear this bit to zero.

This bit is not valid and reads **UNKNOWN** if any of the following are true:

- \$ERR<n>\$STATUS.V is set to 0.
- \$ERR<n>\$STATUS.UE is set to 0 and this bit is never set to 1 by a Deferred error.
- \$ERR<n>\$STATUS.\{UE, DE\} are both set to 0, and this bit can be set to 1 by a Deferred error.

This bit is read/write-one-to-clear.

#### Note

An External abort signaled by the node might be masked and not generate any exception.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

### OF, bit [27]

**Overflow.**

Indicates that multiple errors have been detected. This bit is set to 1 when one of the following occurs:

- A Corrected error counter is implemented, an error is counted, and the counter overflows.
ERR<n>STATUS.V was previously set to 1, a Corrected error counter is not implemented, and a Corrected error is recorded.

ERR<n>STATUS.V was previously set to 1, and a type of error other than a Corrected error is recorded. Otherwise, this bit is unchanged when an error is recorded.

If a Corrected error counter is implemented:

• A direct write that modifies the counter overflow flag indirectly might set this bit to an UNKNOWN value.

• A direct write to this bit that clears this bit to zero might indirectly set the counter overflow flag to an UNKNOWN value.

<table>
<thead>
<tr>
<th>OF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Since this bit was last cleared to zero, no error syndrome has been discarded and, if a Corrected error counter is implemented, it has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Since this bit was last cleared to zero, at least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed.</td>
</tr>
</tbody>
</table>

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<n>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<n>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

The following resets apply:

• This bit is preserved on an Error Recovery reset.

• On a Cold reset, this field resets to an architecturally UNKNOWN value.

MV, bit [26]

Miscellaneous Registers Valid.

<table>
<thead>
<tr>
<th>MV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;MISC0, ERR&lt;n&gt;MISC1, ERR&lt;n&gt;MISC2, and ERR&lt;n&gt;MISC3 are not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The IMPLEMENTATION DEFINED contents of the ERR&lt;n&gt;MISC0, ERR&lt;n&gt;MISC1, ERR&lt;n&gt;MISC2, and ERR&lt;n&gt;MISC3 registers contain additional information for an error recorded by this record.</td>
</tr>
</tbody>
</table>

This bit is read/write-one-to-clear.

Note

If the ERR<n>MISC0, ERR<n>MISC1, ERR<n>MISC2, and ERR<n>MISC3 registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The following resets apply:

• This bit is preserved on an Error Recovery reset.

• On a Cold reset, this field resets to 0.

CE, bits [25:24]

Corrected Error.
The mechanism by which a node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when an error is corrected.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then software must write ones to this field to clear this field to zero.

If ERR<n>STATUS.V is set to 0, this field is not valid and reads UNKNOWN.

This field is read/write-one-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

DE, bit [23]

Deferred Error.

<table>
<thead>
<tr>
<th>DE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No errors were deferred.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one error was not corrected and deferred.</td>
</tr>
</tbody>
</table>

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this bit is nonzero, then software must write 1 to this bit to clear this bit to zero.

If ERR<n>STATUS.V is set to 0, this bit is not valid and reads UNKNOWN.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PN, bit [22]

Poison.

<table>
<thead>
<tr>
<th>PN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC).</td>
</tr>
<tr>
<td>0b1</td>
<td>Uncorrected error or Deferred error recorded because a poison value was detected.</td>
</tr>
</tbody>
</table>

**Note**

- If a producer node detects a corrupt value and defers the error by producing a poison value, then this bit is set to 0 at the producer node.
- This might only be an indication of poison, because, in some EDC schemes, a poison value is encoded as an unlikely form of corrupt data, meaning it is possible to mistake a corrupt value as a poison value.
It is IMPLEMENTATION DEFINED whether a node can distinguish a poison value from a corrupt value.

When clearing ERR<
>STATUS.V to 0, if this bit is nonzero, then software must write 1 to this bit to clear this bit to zero.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<
>STATUS.V is set to 0.
- ERR<
>STATUS.{DE, UE} are both set to 0.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**UET, bits [21:20]**

Uncorrected Error Type.

Describes the state of the component after detecting or consuming an Uncorrected error.

<table>
<thead>
<tr>
<th>UET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Uncorrected error; Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b01</td>
<td>Uncorrected error; Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncorrected error; Latent or Restartable error (UEO).</td>
</tr>
<tr>
<td>0b11</td>
<td>Uncorrected error; Signaled or Recoverable error (UER).</td>
</tr>
</tbody>
</table>

When clearing ERR<
>STATUS.V to 0, if this field is nonzero, then software must write ones to this field to clear this field to zero.

This field is not valid and reads UNKNOWN if any of the following are true:

- ERR<
>STATUS.V is set to 0.
- ERR<
>STATUS.UE is set to 0.

This field is read/write-one-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**CI, bit [19]**

Critical error.

Indicates whether a critical error condition has been recorded.

<table>
<thead>
<tr>
<th>CI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No critical error condition recorded.</td>
</tr>
<tr>
<td>0b1</td>
<td>Critical error condition recorded.</td>
</tr>
</tbody>
</table>

When clearing ERR<
>STATUS.V to 0, if this bit is nonzero, then software must write 1 to this bit to clear this bit to zero.

This bit is not valid and reads UNKNOWN if ERR<
>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
• On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [18:16]**

Reserved, RES0.

**IERR, bits [15:8]**

**IMPLEMENTATION DEFINED** error code.

Used with any primary error code SERR value. Further **IMPLEMENTATION DEFINED** information can be placed in the MISC registers.

This field is not valid and reads **UNKNOWN** if ERR<n>STATUS.V is set to 0.

The following resets apply:

• This field is preserved on an Error Recovery reset.

• On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**SERR, bits [7:0]**

Architecturally-defined primary error code.

Indicates the type of error. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry.
<table>
<thead>
<tr>
<th>SERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>No error.</td>
</tr>
<tr>
<td>0x01</td>
<td>IMPLEMENTATION DEFINED error.</td>
</tr>
<tr>
<td>0x02</td>
<td>Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.</td>
</tr>
<tr>
<td>0x03</td>
<td>IMPLEMENTATION DEFINED pin. For example, nSEI pin.</td>
</tr>
<tr>
<td>0x04</td>
<td>Assertion failure. For example, consistency failure.</td>
</tr>
<tr>
<td>0x05</td>
<td>Error detected on internal data path. For example, parity on ALU result.</td>
</tr>
<tr>
<td>0x06</td>
<td>Data value from associative memory. For example, ECC error on cache data.</td>
</tr>
<tr>
<td>0x07</td>
<td>Address/control value from associative memory. For example, ECC error on cache tag.</td>
</tr>
<tr>
<td>0x08</td>
<td>Data value from a TLB. For example, ECC error on TLB data.</td>
</tr>
<tr>
<td>0x09</td>
<td>Address/control value from a TLB. For example, ECC error on TLB tag.</td>
</tr>
<tr>
<td>0x0A</td>
<td>Data value from producer. For example, parity error on write data bus.</td>
</tr>
<tr>
<td>0x0B</td>
<td>Address/control value from producer. For example, parity error on address bus.</td>
</tr>
<tr>
<td>0x0C</td>
<td>Data value from (non-associative) external memory. For example, ECC error in SDRAM.</td>
</tr>
<tr>
<td>0x0D</td>
<td>Illegal address (software fault). For example, access to unpopulated memory.</td>
</tr>
<tr>
<td>0x0E</td>
<td>Illegal access (software fault). For example, byte write to word register.</td>
</tr>
<tr>
<td>0x0F</td>
<td>Illegal state (software fault). For example, device not ready.</td>
</tr>
<tr>
<td>0x10</td>
<td>Internal data register. For example, parity on a SIMD&amp;FP register. For a PE, all general-purpose, stack pointer, SIMD&amp;FP, and SVE registers are data registers.</td>
</tr>
<tr>
<td>0x11</td>
<td>Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&amp;FP, and SVE registers are control registers.</td>
</tr>
<tr>
<td>0x12</td>
<td>Error response from slave. For example, error response from cache write-back.</td>
</tr>
<tr>
<td>0x13</td>
<td>External timeout. For example, timeout on interaction with another node.</td>
</tr>
<tr>
<td>0x14</td>
<td>Internal timeout. For example, timeout on interface within the node.</td>
</tr>
<tr>
<td>0x15</td>
<td>Deferred error from slave not supported at master. For example, poisoned data received from a slave by a master that cannot defer the error further.</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

This field is not valid and reads UNKNOWN if ERR<n>STATUS.V is set to 0.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

<table>
<thead>
<tr>
<th>Bits [63:32]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
</table>
**AV, bit [31]**

Address Valid.

<table>
<thead>
<tr>
<th>AV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;ADDR not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;ADDR contains an address associated with the highest priority error recorded by this record.</td>
</tr>
</tbody>
</table>

This bit ignores writes if any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to 0.

**V, bit [30]**

Status Register Valid.

<table>
<thead>
<tr>
<th>V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;STATUS not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>ERR&lt;n&gt;STATUS valid. At least one error has been recorded.</td>
</tr>
</tbody>
</table>

This bit ignores writes if any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to 0.

**UE, bit [29]**

Uncorrected error.

<table>
<thead>
<tr>
<th>UE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No errors have been detected, or all detected errors have been either corrected or deferred.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one detected error was not corrected and not deferred.</td>
</tr>
</tbody>
</table>

When clearing ERR<n>STATUS.V to 0, if this bit is nonzero, then software must write one to this bit to clear this bit to zero.

If ERR<n>STATUS.OF is set to 1 and is not being cleared to 0 in the same write, this bit ignores writes.

This bit is not valid and reads UNKNOWN if ERR<n>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**ER, bit [28]**

Error Reported.
### ER, bit [0]

<table>
<thead>
<tr>
<th>ER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No in-band error (External abort) reported.</td>
</tr>
</tbody>
</table>
| 0b1 | An External abort was signaled by the node to the master making the access or other transaction. This can be because any of the following are true:  
  - The applicable one of the ERR<q>CTLR. {WUE, RUE, UE} bits is implemented, and was set to 1 when an Uncorrected error was detected.  
  - The applicable one of the ERR<q>CTLR. {WUE, RUE, UE} bits is not implemented. |

It is IMPLEMENTATION DEFINED whether this bit can be set to 1 by a Deferred error.

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when:

- Clearing ERR<n>STATUS.V to 0.
- Clearing ERR<n>STATUS.UE to 0, if this bit is never set to 1 by a Deferred error.
- Clearing both ERR<n>STATUS.{UE, DE} to 0, if this bit can be set to 1 by a Deferred error.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<n>STATUS.V is set to 0.
- ERR<n>STATUS.UE is set to 0 and this bit is never set to 1 by a Deferred error.
- ERR<n>STATUS.{UE, DE} are both set to 0, and this bit can be set to 1 by a Deferred error.

This bit ignores writes if any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.

#### Note

An External abort signaled by the node might be masked and not generate any exception.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

### OF, bit [27]

**Overflow.**

Indicates that multiple errors have been detected. This bit is set to 1 when one of the following occurs:

- An Uncorrected error is detected and ERR<n>STATUS.UE == 1.
- A Deferred error is detected, ERR<n>STATUS.UE == 0 and ERR<n>STATUS.DE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and ERR<n>STATUS.CE != 0b00. ERR<n>STATUS.CE might be updated for the new Corrected error.
- A Corrected error counter is implemented, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is set to 1 when one of the following occurs:

- A Deferred error is detected and ERR<n>STATUS.UE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, and either or both the ERR<n>STATUS.UE or ERR<n>STATUS.DE bits are set to 1.
A Corrected error counter is implemented, either or both the ERR<n>STATUS.UE or ERR<n>STATUS.DE bits are set to 1, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this bit is set to 0 when one of the following occurs:

- An Uncorrected error is detected and ERR<n>STATUS.UE == 0.
- A Deferred error is detected, ERR<n>STATUS.UE == 0 and ERR<n>STATUS.DE == 0.
- A Corrected error is detected, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0 and ERR<n>STATUS.CE == 0b00.

The IMPLEMENTATION DEFINED clearing of this bit might also depend on the value of the other error status bits.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this bit to an UNKNOWN value.
- A direct write to this bit that clears this bit to 0 might indirectly set the counter overflow flag to an UNKNOWN value.

<table>
<thead>
<tr>
<th>OF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If ERR&lt;n&gt;STATUS.UE == 1, then no error syndrome for an Uncorrected error has been discarded.</td>
</tr>
<tr>
<td></td>
<td>If ERR&lt;n&gt;STATUS.UE == 0 and ERR&lt;n&gt;STATUS.DE == 1, then no error syndrome for a Deferred error has been discarded.</td>
</tr>
<tr>
<td></td>
<td>If ERR&lt;n&gt;STATUS.UE == 0, ERR&lt;n&gt;STATUS.DE == 0, and a Corrected error counter is implemented, then the counter has not overflowed.</td>
</tr>
<tr>
<td></td>
<td>If ERR&lt;n&gt;STATUS.UE == 0, ERR&lt;n&gt;STATUS.DE == 0, ERR&lt;n&gt;STATUS.CE != 0b00, and no Corrected error counter is implemented, then no error syndrome for a Corrected error has been discarded.</td>
</tr>
</tbody>
</table>

**Note**

This bit might have been set to 1 when an error syndrome was discarded and later cleared to 0 when a higher priority syndrome was recorded.

| 0b1  | At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed. |

If this bit is nonzero, then software must write 1 to this bit, to clear this bit to zero, when clearing ERR<n>STATUS.V to 0.

This bit is not valid and reads UNKNOWN if ERR<n>STATUS.V is set to 0.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**MV, bit [26]**

Miscellaneous Registers Valid.

<table>
<thead>
<tr>
<th>MV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ERR&lt;n&gt;MISC0 and ERR&lt;n&gt;MISC1 not valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The IMPLEMENTATION DEFINED contents of the ERR&lt;n&gt;MISC0 and ERR&lt;n&gt;MISC1 registers contains additional information for an error recorded by this record.</td>
</tr>
</tbody>
</table>

This bit ignores writes if any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write.

This bit is read/write-one-to-clear.
If the ERR<n>MISC0 and ERR<n>MISC1 registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to 0.

**CE, bits [25:24]**

Corrected Error.

<table>
<thead>
<tr>
<th>CE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>No errors were corrected.</td>
</tr>
<tr>
<td>0b01</td>
<td>At least one transient error was corrected.</td>
</tr>
<tr>
<td>0b10</td>
<td>At least one error was corrected.</td>
</tr>
<tr>
<td>0b11</td>
<td>At least one persistent error was corrected.</td>
</tr>
</tbody>
</table>

The mechanism by which a node detects whether a correctable error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when an error is corrected.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then software must write ones to this field to clear this field to zero.

If ERR<n>STATUS.OF is set to 1 and is not being cleared to 0 in the same write, this field ignores writes.

If ERR<n>STATUS.V is set to 0, this field is not valid and reads UNKNOWN.

This field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**DE, bit [23]**

Deferred Error.

<table>
<thead>
<tr>
<th>DE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No errors were deferred.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one error was not corrected and deferred.</td>
</tr>
</tbody>
</table>

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this bit is nonzero, then software must write 1 to this bit to clear this bit to zero.

If ERR<n>STATUS.OF is set to 1 and is not being cleared to 0 in the same write, this bit ignores writes.

If ERR<n>STATUS.V is set to 0, this bit is not valid and reads UNKNOWN.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.
**PN, bit [22]**

Poison.

<table>
<thead>
<tr>
<th>PN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC).</td>
</tr>
</tbody>
</table>

**Note**
If a producer node detects a corrupt value and defers the error by producing a poison value, then this bit is set to 0 at the producer node.

<table>
<thead>
<tr>
<th>PN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1</td>
<td>Uncorrected error or Deferred error recorded because a poison value was detected.</td>
</tr>
</tbody>
</table>

**Note**
This might only be an indication of poison, because, in some EDC schemes, a poison value is encoded as an unlikely form of corrupt data, meaning it is possible to mistake a corrupt value as a poison value.

It is IMPLEMENTATION DEFINED whether a node can distinguish a poison value from a corrupt value.

When clearing ERR<n>STATUS.V to 0, if this bit is nonzero, then software must write 1 to this bit to clear this bit to zero.

When clearing both ERR<n>STATUS.{DE, UE} to 0, if this bit is nonzero, then software must write 1 to this bit to clear this bit to zero.

This bit is not valid and reads UNKNOWN if any of the following are true:

- ERR<n>STATUS.V is set to 0.
- ERR<n>STATUS.{DE, UE} are both set to 0.

When any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write, this bit ignores writes.

This bit is read/write-one-to-clear.

The following resets apply:

- This bit is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally UNKNOWN value.

**UET, bits [21:20]**

Uncorrected Error Type.

Describes the state of the component after detecting or consuming an Uncorrected error.

<table>
<thead>
<tr>
<th>UET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Uncorrected error; Uncontainable error (UC).</td>
</tr>
<tr>
<td>0b01</td>
<td>Uncorrected error; Unrecoverable error (UEU).</td>
</tr>
<tr>
<td>0b10</td>
<td>Uncorrected error; Latent or Restartable error (UEO).</td>
</tr>
<tr>
<td>0b11</td>
<td>Uncorrected error; Signaled or Recoverable error (UER).</td>
</tr>
</tbody>
</table>

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then software must write ones to this field to clear this field to zero.

When clearing ERR<n>STATUS.UE to 0, if this field is nonzero, then software must write ones to this field to clear this field to zero.

This field is not valid and reads UNKNOWN if any of the following are true:

- ERR<n>STATUS.V is set to 0.
• ERR<n>STATUS.UE is set to 0.

When any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write, this field ignores writes.

This field is read/write-ones-to-clear. Writing a value other than all-zeros or all-ones sets this field to an UNKNOWN value.

The following resets apply:

• This field is preserved on an Error Recovery reset.
• On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Bits [19:16]**

Reserved, RES0.

**IERR, bits [15:8]**

IMPLEMENTATION DEFINED error code.

Used with any primary error code SERR value. Further IMPLEMENTATION DEFINED information can be placed in the MISC registers.

This field is not valid and reads UNKNOWN if ERR<n>STATUS.V is set to 0.

When any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write, this field ignores writes.

The following resets apply:

• This field is preserved on an Error Recovery reset.
• On a Cold reset, this field resets to an architecturally UNKNOWN value.

**SERR, bits [7:0]**

Architecturally-defined primary error code.

Indicates the type of error. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry.
<table>
<thead>
<tr>
<th>SERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>No error.</td>
</tr>
<tr>
<td>0x01</td>
<td>IMPLEMENTATION DEFINED error.</td>
</tr>
<tr>
<td>0x02</td>
<td>Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer.</td>
</tr>
<tr>
<td>0x03</td>
<td>IMPLEMENTATION DEFINED pin. For example, nSEI pin.</td>
</tr>
<tr>
<td>0x04</td>
<td>Assertion failure. For example, consistency failure.</td>
</tr>
<tr>
<td>0x05</td>
<td>Error detected on internal data path. For example, parity on ALU result.</td>
</tr>
<tr>
<td>0x06</td>
<td>Data value from associative memory. For example, ECC error on cache data.</td>
</tr>
<tr>
<td>0x07</td>
<td>Address/control value from associative memory. For example, ECC error on cache tag.</td>
</tr>
<tr>
<td>0x08</td>
<td>Data value from a TLB. For example, ECC error on TLB data.</td>
</tr>
<tr>
<td>0x09</td>
<td>Address/control value from a TLB. For example, ECC error on TLB tag.</td>
</tr>
<tr>
<td>0x0A</td>
<td>Data value from producer. For example, parity error on write data bus.</td>
</tr>
<tr>
<td>0x0B</td>
<td>Address/control value from producer. For example, parity error on address bus.</td>
</tr>
<tr>
<td>0x0C</td>
<td>Data value from (non-associative) external memory. For example, ECC error in SDRAM.</td>
</tr>
<tr>
<td>0x0D</td>
<td>Illegal address (software fault). For example, access to unpopulated memory.</td>
</tr>
<tr>
<td>0x0E</td>
<td>Illegal access (software fault). For example, byte write to word register.</td>
</tr>
<tr>
<td>0x0F</td>
<td>Illegal state (software fault). For example, device not ready.</td>
</tr>
<tr>
<td>0x10</td>
<td>Internal data register. For example, parity on a SIMD&amp;FP register. For a PE, all general-purpose, stack pointer, and SIMD&amp;FP registers are data registers.</td>
</tr>
<tr>
<td>0x11</td>
<td>Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, and SIMD&amp;FP registers are control registers.</td>
</tr>
<tr>
<td>0x12</td>
<td>Error response from slave. For example, error response from cache write-back.</td>
</tr>
<tr>
<td>0x13</td>
<td>External timeout. For example, timeout on interaction with another node.</td>
</tr>
<tr>
<td>0x14</td>
<td>Internal timeout. For example, timeout on interface within the node.</td>
</tr>
<tr>
<td>0x15</td>
<td>Deferred error from slave not supported at master. For example, poisoned data received from a slave by a master that cannot defer the error further.</td>
</tr>
</tbody>
</table>

All other values are reserved. Reserved values might be defined in a future version of the architecture.

This field is not valid and reads **UNKNOWN** if ERR<n>STATUS.V is set to 0.

When any of ERR<n>STATUS.{CE, DE, UE} are set to 1, and the highest priority of these is not being cleared to 0 in the same write, this field ignores writes.

The following resets apply:

- This field is preserved on an Error Recovery reset.
- On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

## Accessing the ERR<n>STATUS

After reading the status register, software must clear the valid bits to allow new errors to be recorded.

Between reading the register and clearing the valid bits, a new error might have overwritten the register. To prevent this new error being lost:

- When ARMv8.4-RAS is not implemented, some control bits use a form of read/write-one-to-clear and ignore writes depending on the values of other bits and which bits are being cleared.
- When ARMv8.4-RAS is implemented, a write to ERR<n>STATUS is ignored if both:
  - Any of the ERR<n>STATUS.{V, UE, OF, CE, DE} fields are nonzero before the write.
The write does not clear the nonzero ERR<n>STATUS.V, UE, OF, CE, DE} field(s) to zero by writing one(s) to the applicable field(s).

Software must write ones to the {ER, PN, UET, CI} fields when clearing ERR<n>STATUS.V, UE, OF, CE, DE}.

**ERR<n>STATUS can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0x010 + 64n</td>
<td>ERR&lt;n&gt;STATUS</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The ERRPIDR0 characteristics are:

**Purpose**

Provides discovery information about the component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

Implementation of this register is OPTIONAL.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRPIDR0 are RES0.

**Attributes**

ERRPIDR0 is a 32-bit register.

**Field descriptions**

The ERRPIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>29-10</td>
<td>PART_0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PART_0, bits [7:0]**

Part number, bits [7:0].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number, and:

- If a 12-bit part number is used, it is stored in ERRPIDR1.PART_1 and this field.
- If a 16-bit part number is used, it is stored in ERRPIDR2.PART_2, ERRPIDR1.PART_1 and this field.

This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the ERRPIDR0**

**ERRPIDR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFE0</td>
<td>ERRPIDR0</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

**Purpose**

Provides discovery information about the component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

Implementation of this register is OPTIONAL.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRPIDR1 are RES0.

**Attributes**

ERRPIDR1 is a 32-bit register.

**Field descriptions**

The ERRPIDR1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | DES_0 | PART_1 |

**Bits [31:8]**

Reserved, RES0.

**DES_0, bits [7:4]**

Designer JEP106 identification code, bits [3:0]. This field and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component.

The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

**Note**

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field reads as an IMPLEMENTATION DEFINED value.

**PART_1, bits [3:0]**

Part number, bits [11:8]

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number, and:

- If a 12-bit part number is used, it is stored in this field and ERRPIDR0.PART_0.
- If a 16-bit part number is used, it is stored in ERRPIDR2.PART_2, this field, and ERRPIDR0.PART_0.
This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the ERRPIDR1**

**ERRPIDR1 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFE4</td>
<td>ERRPIDR1</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
The ERRPIDR2 characteristics are:

**Purpose**

Provides discovery information about the component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

Implementation of this register is **OPTIONAL**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRPIDR2 are **RES0**.

**Attributes**

ERRPIDR2 is a 32-bit register.

**Field descriptions**

The ERRPIDR2 bit assignments are:

**When the component uses a 12-bit part number:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>REVISION</td>
</tr>
<tr>
<td>28</td>
<td>JEDEC</td>
</tr>
<tr>
<td>27</td>
<td>DES_1</td>
</tr>
<tr>
<td>26</td>
<td>REVAND</td>
</tr>
<tr>
<td>25</td>
<td>Component major revision.</td>
</tr>
<tr>
<td>24</td>
<td>This field and ERRPIDR3.REVAND together form the revision number of the component, with REVISION being the most significant part and REVAND the least significant part.</td>
</tr>
<tr>
<td>23</td>
<td>This field reads as an IMPLEMENTATION DEFINED value.</td>
</tr>
<tr>
<td>22</td>
<td>JEDEC-assigned JEP106 implementer code is used. This bit is RAO.</td>
</tr>
<tr>
<td>21</td>
<td>Designer, JEP106 identification code, bits [6:4]. ERRPIDR1_DES_0 and this field together form the JEDEC-assigned JEP106 identification code for the designer of the component.</td>
</tr>
<tr>
<td>20</td>
<td>This field reads as an IMPLEMENTATION DEFINED value.</td>
</tr>
</tbody>
</table>

**Note**
For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

When the component uses a 16-bit part number:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30-24</td>
<td>PART_2, bits [7:4]. Part number, bits [15:12]. The part number is selected by the designer of the component. It is stored in this field, <code>ERRPIDR1.PART_1</code> and <code>ERRPIDR0.PART_0</code>. This field reads as an IMPLEMENTATION DEFINED value.</td>
</tr>
<tr>
<td>23-20</td>
<td>JEDEC, bit [3]. JEDEC-assigned JEP106 implementer code is used. This bit is RAO.</td>
</tr>
<tr>
<td>19-16</td>
<td>DES_1, bits [2:0]. Designer, JEP106 identification code, bits [6:4]. <code>ERRPIDR1.DES_0</code> and this field together form the JEDEC-assigned JEP106 identification code for the designer of the component. This field reads as an IMPLEMENTATION DEFINED value.</td>
</tr>
</tbody>
</table>

Accessing the ERRPIDR2

**ERRPIDR2 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFE8</td>
<td>ERRPIDR2</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

**Purpose**

Provides discovery information about the component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

Implementation of this register is **OPTIONAL**.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRPIDR3 are **RES0**.

**Attributes**

ERRPIDR3 is a 32-bit register.

**Field descriptions**

The ERRPIDR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | REVAND | CMOD |

**Bits [31:8]**

Reserved, RES0.

**REVAND, bits [7:4]**

*When the component uses a 12-bit part number:*

Component minor revision. ERRPIDR2.REVISION and this field together form the revision number of the component, with REVISION being the most significant part and REVAND the least significant part.

This field reads as an **IMPLEMENTATION DEFINED** value.

*When the component uses a 16-bit part number:*

Component revision.

This field reads as an **IMPLEMENTATION DEFINED** value.

**Otherwise:**

Reserved, RES0.

**CMOD, bits [3:0]**

Customer Modified. If the value of this field is non-zero, then the component has **IMPLEMENTATION DEFINED** modifications.
<table>
<thead>
<tr>
<th>CMOD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The component is not modified from the original design.</td>
</tr>
</tbody>
</table>

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.

This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the ERRPIDR3**

**ERRPIDR3 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFEC</td>
<td>ERRPIDR3</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

**Purpose**

Provides discovery information about the component.

For more information, see About the Peripheral identification scheme in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**Configuration**

Implementation of this register is OPTIONAL.

This register is present only when RAS is implemented. Otherwise, direct accesses to ERRPIDR4 are RES0.

**Attributes**

ERRPIDR4 is a 32-bit register.

**Field descriptions**

The ERRPIDR4 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7:4</td>
<td>SIZE</td>
</tr>
<tr>
<td>3:0</td>
<td>DES_2</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SIZE, bits [7:4]**

Size of the component. The distance from the start of the address space used by this component to the end of the component identification registers.

If the value of this field is non-zero, then the component occupies $2^{\text{ERRPIDR4.SIZE}}$ 4KB blocks.

<table>
<thead>
<tr>
<th>SIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>One of the following is true:</td>
</tr>
<tr>
<td></td>
<td>• The component uses a single 4KB block.</td>
</tr>
<tr>
<td></td>
<td>• The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.</td>
</tr>
</tbody>
</table>

Using this bit field to indicate the size of the component is deprecated. This bit field might not correctly indicate the size of the component.

Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This register reads as an IMPLEMENTATION DEFINED value.

**DES_2, bits [3:0]**

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1.
The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component.

This field reads as an IMPLEMENTATION DEFINED value.

**Note**

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value 0x4.

### Accessing the ERRPIDR4

**ERRPIDR4 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS</td>
<td>0xFD0</td>
<td>ERRPIDR4</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
GICC_ABPR, CPU Interface Aliased Binary Point Register

The GICC_ABPR characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

In systems that support two Security states:

- This register is an alias of the Non-secure copy of GICC_BPR.
- Non-secure accesses to this register return a shifted value of the binary point.
- If ICC_CTLR_EL3.CBPR_EL1NS == 1, Secure accesses to this register access ICC_BPR0_EL1.

**Attributes**

The reset value of this register is defined as (minimum GICC_BPR.Binary_Point + 1), resulting in a permitted range of 0x1-0x4.

**Field descriptions**

The GICC_ABPR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

**Bits [31:3]**

Reserved, RES0.

**Binary_Point, bits [2:0]**

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The following list describes how this field determines the interrupt priority bits assigned to the group priority field:

- Priority grouping for Group 1 interrupts when CBPR==0, for the processing of Group 1 interrupts in a GIC implementation that supports interrupt grouping, when GICC_CTLR.CBPR == 0.
- Priority grouping for Group 0 interrupts, or Group 1 interrupts when CBPR==1, for all other cases.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the GICC_ABPR**

This register is used only when System register access is not enabled. When System register access is enabled, the System registers ICC_BPR0_EL1 and ICC_BPR1_EL1 provide equivalent functionality.

**GICC_ABPR can be accessed through the memory-mapped interfaces:**
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICC_AEOIR characteristics are:

**Purpose**

A write to this register performs priority drop for the specified Group 1 interrupt and, if the appropriate GICC_CTLR.EOImodeS or GICC_CTLR.EOImodeNS field == 0, also deactivates the interrupt.

**Configuration**

When GICD_CTLR.DS==0, this register is an alias of the Non-secure view of GICC_EOIR. A Secure access to this register is identical to a Non-secure access to GICC_EOIR.

**Attributes**

GICC_AEOIR is a 32-bit register.

**Field descriptions**

The GICC_AEOIR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

**Accessing the GICC_AEOIR**

A write to this register must correspond to the most recently acknowledged Group 1 interrupt. If a value other than the last value read from GICC_AIAR is written to this register, the effect is UNPREDICTABLE.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR1 provides equivalent functionality.
- For AArch64 implementations, ICC_EOIR1_EL1 provides equivalent functionality.
When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

**GICC_AEOIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0024</td>
<td>GICC_AEOIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
The GICC_AHPPIR characteristics are:

**Purpose**

If the highest priority pending interrupt is in Group 1, this register provides the INTID of the highest priority pending interrupt on the CPU interface.

**Configuration**

If $\text{GICD\_CTLR}.\text{DS}==0$, this register is an alias of the Non-secure view of $\text{GICC\_HPPIR}$. A Secure access to this register is identical to a Non-secure access to $\text{GICC\_HPPIR}$.

**Attributes**

GICC_AHPPIR is a 32-bit register.

**Field descriptions**

The GICC_AHPPIR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

**Accessing the GICC_AHPPIR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, $\text{ICC\_HPPIR1}$ provides equivalent functionality.
- For AArch64 implementations, $\text{ICC\_HPPIR1\_EL1}$ provides equivalent functionality.

If the highest priority pending interrupt is in Group 0, a read of this register returns the special INTID 1023.

Interrupt identifiers corresponding to an interrupt group that is not enabled are ignored.
If the highest priority pending interrupt is a direct interrupt that is both individually enabled in the Distributor and part of an interrupt group that is enabled in the Distributor, and the interrupt group is disabled in the CPU interface for this PE, this register returns the special INTID 1023.

See Preemption for more information about pending interrupts that are not considered when determining the highest priority pending interrupt.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

**GICC_AHPPIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0028</td>
<td>GICC_AHPPIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The GICC_AIAR characteristics are:

**Purpose**

Provides the INTID of the signaled Group 1 interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

**Configuration**

When \texttt{GICD_CTLR.DS==0}, this register is an alias of the Non-secure view of \texttt{GICC_IAR}. A Secure access to this register is identical to a Non-secure access to \texttt{GICC_IAR}.

**Attributes**

GICC_AIAR is a 32-bit register.

**Field descriptions**

The GICC_AIAR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

**Accessing the GICC_AIAR**

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

**GICC_AIAR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GICC_AIAR, CPU Interface Aliased Interrupt Acknowledge Register

<table>
<thead>
<tr>
<th>GIC CPU interface</th>
<th>0x0020</th>
<th>GICC_AIAR</th>
</tr>
</thead>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.
The GICC_APR<n> characteristics are:

**Purpose**

Provides information about interrupt active priorities.

**Configuration**

Some or all RW fields of this register have defined reset values.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

When GICD_CTLR.DS == 0, these registers are Banked, and Non-secure accesses do not affect Secure operation. The Secure copies of these registers hold active priorities for Group 0 interrupts, and the Non-secure copies provide a Non-secure view of the active priorities for Group 1 interrupts.

GICC_APR1 is only implemented in implementations that support 6 or more bits of priority. GICC_APR2 and GICC_APR3 are only implemented in implementations that support 7 bits of priority.

When GICD_CTLR.DS==1, these registers hold the active priorities for Group 0 interrupts, and the active priorities for Group 1 interrupts are held by the GICC_NSAPR<n> registers.

**Attributes**

GICC_APR<n> is a 32-bit register.

**Field descriptions**

The GICC_APR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| IMPLEMENTATION DEFINED |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to 0.

**Accessing the GICC_APR<n>**

These registers are used only when System register access is not enabled. When System register access is enabled the following registers provide equivalent functionality:

- In AArch64:
  - For Group 0, ICC_AP0R<n> EL1.
  - For Group 1, ICC_AP1R<n> EL1.
- In AArch32:
  - For Group 0, ICC_AP0R<n>.
  - For Group 1, ICC_AP1R<n>.
**GICC APR<\(n>\)** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0000 + 4(n)</td>
<td>GICC APR&lt;(n&gt;)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICC_BPR, CPU Interface Binary Point Register

The GICC_BPR characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

In systems that support two Security states:

- This register is Banked.
- The Secure instance of this register determines Group 0 interrupt preemption.
- The Non-secure instance of this register determines Group 1 interrupt preemption.

In systems that support only one Security state, when **GICC_CTLR**.CBPR == 0, this register determines only Group 0 interrupt preemption.

When **GICC_CTLR**.CBPR == 1, this register determines interrupt preemption for both Group 0 and Group 1 interrupts.

**Attributes**

GICC_BPR is a 32-bit register.

**Field descriptions**

The GICC_BPR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Binary_Point</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**Binary_Point, bits [2:0]**

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The following list describes how this field determines the interrupt priority bits assigned to the group priority field:

- Priority grouping for Group 1 interrupts when CBPR==0, for the processing of Group 1 interrupts in a GIC implementation that supports interrupt grouping, when **GICC_CTLR**.CBPR == 0.
- Priority grouping for Group 0 interrupts, or Group 1 interrupts when CBPR==1, for all other cases.

This field resets to an architecturally **UNKNOWN** value.

**Note**

Aliasing the Non-secure GICC_BPR as **GICC_ABPR** in a multiprocessor system permits a PE that can make only Secure accesses to configure the preemption setting for Group 1 interrupts by accessing **GICC_ABPR**.
Accessing the GICC_BPR

This register is used only when System register access is not enabled. When System register access is enabled this register is RAZ/WI, and the System registers ICC_BPR0_EL1 and ICC_BPR1_EL1 provide equivalent functionality.

GICC_BPR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0008</td>
<td>GICC_BPR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICC_CTLR characteristics are:

**Purpose**

Controls the CPU interface, including enabling of interrupt groups, interrupt signal bypass, binary point registers used, and separation of priority drop and interrupt deactivation.

**Note**

If the GIC implementation supports two Security states, independent EOI controls are provided for accesses from each Security state. Secure accesses handle both Group 0 and Group 1 interrupts, and Non-secure accesses handle Group 1 interrupts only.

**Configuration**

Some or all RW fields of this register have defined reset values.

In a GIC implementation that supports two Security states:

- This register is Banked.
- The register bit assignments are different in the Secure and Non-secure copies.

**Attributes**

GICC_CTLR is a 32-bit register.

**Field descriptions**

The GICC_CTLR bit assignments are:

**When GICD_CTLR.DS==0, Non-secure access:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>EOImodeNS</td>
<td>RES0</td>
<td>RQBypDisGrp1</td>
<td>FIQBypDisGrp1</td>
<td>RES0</td>
<td>EnableGrp1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:10]**

Reserved, RES0.

**EOImodeNS, bit [9]**

Controls the behavior of Non-secure accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

<table>
<thead>
<tr>
<th>EOImodeNS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

**Note**
An implementation is permitted to make this bit RAO/WI.

This field resets to 0.

Bits [8:7]

Reserved, RES0.

IRQBypDisGrp1, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

<table>
<thead>
<tr>
<th>IRQBypDisGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass IRQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass IRQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.
If System register access is enabled for EL1, this field is ignored.
If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.
See Interrupt signal bypass and bypass disable for more information.
This field resets to 0.

FIQBypDisGrp1, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

<table>
<thead>
<tr>
<th>FIQBypDisGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass FIQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass FIQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DFB == 1, this field is RAO/WI.
If System register access is enabled for EL1, this field is ignored.
If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.
See Interrupt signal bypass and bypass disable for more information.
This field resets to 0.

Bits [4:1]

Reserved, RES0.

EnableGrp1, bit [0]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

<table>
<thead>
<tr>
<th>EnableGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 interrupt signaling is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 interrupt signaling is enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.
When GICD_CTLR.DS==0, Secure access:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>EOImodeNS</td>
<td>EOImodeS</td>
<td>IRQBypDisGrp1</td>
<td>FIQBypDisGrp1</td>
<td>IRQBypDisGrp0</td>
<td>FIQBypDisGrp0</td>
<td>CBPR</td>
<td>FIQEn</td>
<td>RES0</td>
<td>EnableGrp1</td>
<td>EnableGrp0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [31:11]

Reserved, RES0.

EOImodeNS, bit [10]

Controls the behavior of Non-secure accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

<table>
<thead>
<tr>
<th>EOImodeNS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

Note

An implementation is permitted to make this bit RAO/WI.

This field resets to 0.

EOImodeS, bit [9]

Controls the behavior of Secure accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

<table>
<thead>
<tr>
<th>EOImodeS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

Note

An implementation is permitted to make this bit RAO/WI.

This field shares state with GICC_CTLR.EOImode.

This field resets to 0.

IRQBypDisGrp1, bit [8]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

<table>
<thead>
<tr>
<th>IRQBypDisGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass IRQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass IRQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.
This field resets to 0.

FIQBypDisGrp1, bit [7]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

<table>
<thead>
<tr>
<th>FIQBypDisGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass FIQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass FIQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DFB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

IRQBypDisGrp0, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 0:

<table>
<thead>
<tr>
<th>IRQBypDisGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass IRQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass IRQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

FIQBypDisGrp0, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 0:

<table>
<thead>
<tr>
<th>FIQBypDisGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass FIQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass FIQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

CBPR, bit [4]

Controls whether GICC_BPR provides common control of preemption to Group 0 and Group 1 interrupts:
CBPR

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICC_BPR determines preemption for Group 0 interrupts only.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICC_BPR determines preemption for both Group 0 and Group 1 interrupts.</td>
</tr>
</tbody>
</table>

This field is an alias of ICC_CTLR_EL3.CBPR_EL1NS.

In a GIC that supports two Security states, when CBPR == 1:

- A Non-secure read of GICC_BPR returns the value of Secure GICC_BPR.BinaryPoint, incremented by 1, and saturated to 0b111.
- Non-secure writes of GICC_BPR are ignored.

This field resets to 0.

**FIQEn, bit [3]**

Controls whether the CPU interface signals Group 0 interrupts to a target PE using the FIQ or IRQ signal:

<table>
<thead>
<tr>
<th>FIQEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupts are signaled using the IRQ signal.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupts are signaled using the FIQ signal.</td>
</tr>
</tbody>
</table>

Group 1 interrupts are signaled using the IRQ signal only.

If an implementation supports two Security states, this bit is permitted to be RAO/WI.

This field resets to 0.

**Bit [2]**

Reserved, RES0.

**EnableGrp1, bit [1]**

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

<table>
<thead>
<tr>
<th>EnableGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 interrupt signaling is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 interrupt signaling is enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**EnableGrp0, bit [0]**

Enables the signaling of Group 0 interrupts by the CPU interface to a target PE:

<table>
<thead>
<tr>
<th>EnableGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupt signaling is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupt signaling is enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**When GICD_CTLR.DS == 0b1:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>EOlmode</td>
</tr>
<tr>
<td>RQByDisGrp1</td>
<td>FIQBypDisGrp1</td>
</tr>
<tr>
<td>RQByDisGrp0</td>
<td>FIQBypDisGrp0</td>
</tr>
</tbody>
</table>

**Bits [31:10]**

Reserved, RES0.
EOImode, bit [9]

Controls the behavior of accesses to GICC_EOIR, GICC_AEOIR, and GICC_DIR.

<table>
<thead>
<tr>
<th>EOImode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality.</td>
</tr>
</tbody>
</table>

Note

An implementation is permitted to make this bit RAO/WI.

This field shares state with GICC_CTLR.EOImodeS.

This field resets to 0.

IRQBypDisGrp1, bit [8]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

<table>
<thead>
<tr>
<th>IRQBypDisGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass IRQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass IRQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

FIQBypDisGrp1, bit [7]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

<table>
<thead>
<tr>
<th>FIQBypDisGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass FIQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass FIQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DFB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

IRQBypDisGrp0, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 0:

<table>
<thead>
<tr>
<th>IRQBypDisGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass IRQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass IRQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>
If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

**FIQBypDisGrp0, bit [5]**

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 0:

<table>
<thead>
<tr>
<th>FIQBypDisGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bypass FIQ signal is signaled to the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bypass FIQ signal is not signaled to the PE.</td>
</tr>
</tbody>
</table>

If System register access is enabled for EL3 and ICC_SRE_EL3.DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

See Interrupt signal bypass and bypass disable for more information.

This field resets to 0.

**CBPR, bit [4]**

Controls whether GICC_BPR provides common control of preemption to Group 0 and Group 1 interrupts:

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICC_BPR determines preemption for Group 0 interrupts only.</td>
</tr>
<tr>
<td></td>
<td>GICC_ABPR determines preemption for Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICC_BPR determines preemption for both Group 0 and Group 1 interrupts.</td>
</tr>
</tbody>
</table>

This field is an alias of ICC_CTLR_EL3 CBPR_EL1NS.

In a GIC that supports two Security states, when CBPR == 1:

- A Non-secure read of GICC_BPR returns the value of Secure GICC_BPR.BinaryPoint, incremented by 1, and saturated to 0b111.
- Non-secure writes of GICC_BPR are ignored.

This field resets to 0.

**FIQEn, bit [3]**

Controls whether the CPU interface signals Group 0 interrupts to a target PE using the FIQ or IRQ signal:

<table>
<thead>
<tr>
<th>FIQEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupts are signaled using the IRQ signal.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupts are signaled using the FIQ signal.</td>
</tr>
</tbody>
</table>

Group 1 interrupts are signaled using the IRQ signal only.

If an implementation supports two Security states, this bit is permitted to be RAO/WI.

This field resets to 0.

**Bit [2]**

Reserved, RES0.
EnableGrp1, bit [1]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

<table>
<thead>
<tr>
<th>EnableGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 interrupt signaling is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 interrupt signaling is enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

EnableGrp0, bit [0]

Enables the signaling of Group 0 interrupts by the CPU interface to a target PE:

<table>
<thead>
<tr>
<th>EnableGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupt signaling is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupt signaling is enabled.</td>
</tr>
</tbody>
</table>

This field resets to 0.

Accessing the GICC_CTLR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_CTLR and ICC_MCTLR provide equivalent functionality.
- For AArch64 implementations, ICC_CTLR_EL1 and ICC_CTLR_EL3 provide equivalent functionality.

GICC_CTLR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0000</td>
<td>GICC_CTLR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GICC_DIR, CPU Interface Deactivate Interrupt Register

The GICC_DIR characteristics are:

**Purpose**

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

**Configuration**

**Attributes**

GICC_DIR is a 32-bit register.

**Field descriptions**

The GICC_DIR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

**Accessing the GICC_DIR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_DIR provides equivalent functionality.
- For AArch64 implementations, ICC_DIR_EL1 provides equivalent functionality.

Writes to this register have an effect only in the following cases:

- When GICD_CTLR.DS == 1, if GICC_CTLR.EOImode == 1.
- In GIC implementations that support two Security states:
  - If the access is Secure and GICC_CTLR.EOImodeS == 1.
  - If the access is Non-secure and GICC_CTLR.EOImodeNS == 1.

The following writes must be ignored:
• Writes to this register when the corresponding EOImode field in \texttt{GICC_CTLR} == 0. In systems that support system error generation, an implementation might generate a system error.
• Writes to this register when the corresponding EOImode field in \texttt{GICC_CTLR} == 0 and the corresponding interrupt is not active. In systems that support system error generation, an implementation might generate a system error. In implementations using the GIC Stream Protocol Interface these writes correspond to a Deactivate packet for an interrupt that is not active.

If the corresponding EOImode field in \texttt{GICC_CTLR} is 1 and this register is written to without a corresponding write to \texttt{GICC_EOIR} or \texttt{GICC_AEOIR}, the interrupt is deactivated but the bit corresponding to it in the active priorities registers remains set.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

\textbf{GICC\_DIR can be accessed through the memory-mapped interfaces:}

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x1000</td>
<td>GICC_DIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When GICD\_CTRL\_DS == 0b0 accesses to this register are \texttt{WO}.
• When IsAccessSecure() accesses to this register are \texttt{WO}.
• When !IsAccessSecure() accesses to this register are \texttt{WO}.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICC_EOIR, CPU Interface End Of Interrupt Register

The GICC_EOIR characteristics are:

**Purpose**

A write to this register performs priority drop for the specified interrupt and, if the appropriate `GICC_CTLR.EOImodeS` or `GICC_CTLR.EOImodeNS` field == 0, also deactivates the interrupt.

**Configuration**

If `GICD_CTLR.DS==0`:

- This register is Common.
- `GICC_AEOIR` is an alias of the Non-secure view of this register.

For Secure writes when `GICD_CTLR.DS==0`, or for Secure and Non-secure writes when `GICD_CTLR.DS==1`, the register provides functionality for Group 0 interrupts.

For Non-secure writes when `GICD_CTLR.DS==1`, the register provides functionality for Group 1 interrupts.

**Attributes**

GICC_EOIR is a 32-bit register.

**Field descriptions**

The GICC_EOIR bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| RES0   | INTID  |

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

For every read of a valid INTID from `GICC_IAR`, the connected PE must perform a matching write to GICC_EOIR. The value written to GICC_EOIR must be the INTID from `GICC_IAR`. Reads of INTIDs 1020-1023 do not require matching writes.

**Note**
Arm recommends that software preserves the entire register value read from GICC_IAR, and writes that value back to GICC_EOIR on completion of interrupt processing.

For nested interrupts, the order of writes to this register must be the reverse of the order of interrupt acknowledgement. Behavior is UNPREDICTABLE if:

- This ordering constraint is not maintained.
- The value written to this register does not match an active interrupt, or the ID of a spurious interrupt.
- The value written to this register does not match the last valid interrupt value read from GICC_IAR.

See Interrupt lifecycle for general information about the effect of writes to end of interrupt registers, and about the possible separation of the priority drop and interrupt deactivate operations.

If GICD_CTLR.DS==0:

- GICC_CTLR.EOImodeS controls the behavior of Secure accesses to GICC_EOIR and GICC_AEOIR.
- GICC_CTLR.EOImodeNS controls the behavior of Non-secure accesses to GICC_EOIR and GICC_AEOIR.

### Accessing the GICC_EOIR

The following writes must be ignored:

- Writes of INTIDs 1020-1023.
- Secure writes corresponding to Group 1 interrupts. In systems that support system error generation, an implementation might generate a system error. In this case, GIC behavior is predictable, and the highest Secure active priority (in the Secure copy of GICC_APR<n>) will be reset if the highest active priority is Secure. System behavior is UNPREDICTABLE.
- Non-secure writes corresponding to Group 0 interrupts when GICC_CTLR.EOImodeS == 1. In systems that support system error generation, an implementation might generate a system error. In this case, GIC behavior is predictable, and the highest Non-secure active priority (in the Non-secure copy of GICC_APR<n>) will be reset if the highest active priority is Non-secure. System behavior is UNPREDICTABLE.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR0 and ICC_EOIR1 provide equivalent functionality.
- For AArch64 implementations, ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

### GICC_EOIR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU</td>
<td>0x0010</td>
<td>GICC_EOIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are WO.
- When IsAccessSecure() accesses to this register are WO.
- When !IsAccessSecure() accesses to this register are WO.
The GICC_HPPIR characteristics are:

**Purpose**

Provides the INTID of the highest priority pending interrupt on the CPU interface.

**Configuration**

If \( \text{GICD_CTLR.DS} = 0 \):
- This register is Common.
- \( \text{GICC_AHPPIR} \) is an alias of the Non-secure view of this register.

**Attributes**

GICC_HPPIR is a 32-bit register.

**Field descriptions**

The GICC_HPPIR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>INTID</td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:
- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

**Accessing the GICC_HPPIR**

This register is used only when System register access is not enabled. When System register access is enabled:
- For AArch32 implementations, \( \text{ICC_HPPIR0} \) and \( \text{ICC_HPPIR1} \) provide equivalent functionality.
- For AArch64 implementations, \( \text{ICC_HPPIR0_EL1} \) and \( \text{ICC_HPPIR1_EL1} \) provide equivalent functionality.

If the highest priority pending interrupt is in Group 0, a Non-secure read of this register returns the special INTID 1023.
For Secure reads when `GICD_CTLR.DS==0`, or for Secure and Non-secure reads when `GICD_CTLR.DS==1`, returns the special INTID 1022 if the highest priority pending interrupt is in Group 1.

If no interrupts are in the pending state, a read of this register returns the special INTID 1023.

Interrupt identifiers corresponding to an interrupt group that is not enabled are ignored.

If the highest priority pending interrupt is a direct interrupt that is both individually enabled in the Distributor and part of an interrupt group that is enabled in the Distributor, and the interrupt group is disabled in the CPU interface for this PE, this register returns the special INTID 1023.

See Preemption for more information about pending interrupts that are not considered when determining the highest priority pending interrupt.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

**GICC_HPPIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0018</td>
<td>GICC_HPPIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are RO.
- When `IsAccessSecure()` accesses to this register are RO.
- When `!IsAccessSecure()` accesses to this register are RO.
The GICC_IAR characteristics are:

**Purpose**

Provides the INTID of the signaled interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

**Configuration**

This register is available in all configurations of the GIC. If `GICD_CTLR.DS==0`:

- This register is Common.
- `GICC_AIAR` is an alias of the Non-secure view of this register.

The format of the INTID is governed by whether affinity routing is enabled for a Security state.

**Attributes**

GICC_IAR is a 32-bit register.

**Field descriptions**

The GICC_IAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>...</td>
</tr>
<tr>
<td>29</td>
<td>...</td>
</tr>
<tr>
<td>28</td>
<td>...</td>
</tr>
<tr>
<td>27</td>
<td>...</td>
</tr>
<tr>
<td>26</td>
<td>...</td>
</tr>
<tr>
<td>25</td>
<td>...</td>
</tr>
<tr>
<td>24</td>
<td>...</td>
</tr>
<tr>
<td>23</td>
<td>...</td>
</tr>
<tr>
<td>22</td>
<td>...</td>
</tr>
<tr>
<td>21</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>...</td>
</tr>
<tr>
<td>19</td>
<td>...</td>
</tr>
<tr>
<td>18</td>
<td>...</td>
</tr>
<tr>
<td>17</td>
<td>...</td>
</tr>
<tr>
<td>16</td>
<td>...</td>
</tr>
<tr>
<td>15</td>
<td>...</td>
</tr>
<tr>
<td>14</td>
<td>...</td>
</tr>
<tr>
<td>13</td>
<td>...</td>
</tr>
<tr>
<td>12</td>
<td>...</td>
</tr>
<tr>
<td>11</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>...</td>
</tr>
<tr>
<td>9</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>...</td>
</tr>
<tr>
<td>7</td>
<td>...</td>
</tr>
<tr>
<td>6</td>
<td>...</td>
</tr>
<tr>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
</tr>
<tr>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**INTID, bits [23:0]**

The INTID of the signaled interrupt.

---

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

---

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

A read of this register returns the INTID of the highest priority pending interrupt for the CPU interface. The read returns a spurious INTID of 1023 if any of the following apply:

- Forwarding of interrupts by the Distributor to the CPU interface is disabled.
- Signaling of interrupts by the CPU interface to the connected PE is disabled.
- There are no pending interrupts on the CPU interface with sufficient priority for the interface to signal it to the PE.
When the GIC returns a valid INTID to a read of this register it treats the read as an acknowledge of that interrupt. In addition, it changes the interrupt status from pending to active, or to active and pending if the pending state of the interrupt persists. Normally, the pending state of an interrupt persists only if the interrupt is level-sensitive and remains asserted.

For every read of a valid INTID from GICC_IAR, the connected PE must perform a matching write to GICC_EOIR.

**Note**
- Arm recommends that software preserves the entire register value read from this register, and writes that value back to GICC_EOIR on completion of interrupt processing.
- For SPIs, although multiple target PEs might attempt to read this register at any time, only one PE can obtain a valid INTID. See 'Interrupt acknowledgement', section 4.7.1 of the GICv3 Architecture Specification, for more information.

### Accessing the GICC_IAR

When GICD_CTLR.DS==1, if the highest priority pending interrupt is in Group 1, the special INTID 1022 is returned.

In GIC implementations that support two Security states, if the highest priority pending interrupt is in Group 0, Non-secure reads return the special INTID 1023.

In GIC implementations that support two Security states, if the highest priority pending interrupt is in Group 1, Secure reads return the special INTID 1022.

This register is used only when System register access is not enabled. When System register access is enabled:
- For AArch32 implementations, ICC_IAR0 and ICC_IAR1 provide equivalent functionality.
- For AArch64 implementations, ICC_IAR0_EL1 and ICC_IAR1_EL1 provide equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

**GICC_IAR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x000C</td>
<td>GICC_IAR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GICC_IIDR, CPU Interface Identification Register

The GICC_IIDR characteristics are:

**Purpose**

Provides information about the implementer and revision of the CPU interface.

**Configuration**

**Attributes**

GICC_IIDR is a 32-bit register.

**Field descriptions**

The GICC_IIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ProductID</td>
<td>An implementation defined product identifier.</td>
</tr>
<tr>
<td>19</td>
<td>Architecture_version</td>
<td>The version of the GIC architecture that is implemented.</td>
</tr>
<tr>
<td>11</td>
<td>Revision</td>
<td>An implementation defined revision number for the CPU interface.</td>
</tr>
<tr>
<td>0</td>
<td>Implementer</td>
<td>Contains the JEP106 code of the company that implemented the CPU interface.</td>
</tr>
</tbody>
</table>

**ProductID, bits [31:20]**

An implementation defined product identifier.

**Architecture_version, bits [19:16]**

The version of the GIC architecture that is implemented.

<table>
<thead>
<tr>
<th>Architecture_version</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>GICv1.</td>
</tr>
<tr>
<td>0b0010</td>
<td>GICv2.</td>
</tr>
<tr>
<td>0b0011</td>
<td>GICv3 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.</td>
</tr>
<tr>
<td>0b0100</td>
<td>GICv4 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.</td>
</tr>
</tbody>
</table>

Other values are reserved.

**Revision, bits [15:12]**

An implementation defined revision number for the CPU interface.

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the CPU interface.

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.
Accessing the GICC_IIDR

GICC_IIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x00FC</td>
<td>GICC_IIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRL.RS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The GICC_NSAPR<n> characteristics are:

**Purpose**

Provides information about Group 1 interrupt active priorities.

**Configuration**

Some or all RW fields of this register have defined reset values.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

When GICD_CTLR.DS==0, these registers are RAZ/WI to Non-secure accesses.

GICC_NSAPR1 is only implemented in implementations that support 6 or more bits of priority. GICC_NSAPR2 and GICC_NSAPR3 are only implemented in implementations that support 7 bits of priority.

**Attributes**

GICC_NSAPR<n> is a 32-bit register.

**Field descriptions**

The GICC_NSAPR<n> bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9  | Bit 8  | Bit 7  | Bit 6  | Bit 5  | Bit 4  | Bit 3  | Bit 2  | Bit 1  | Bit 0  |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        | IMPLEMENTATION DEFINED |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field resets to 0.

**Accessing the GICC_NSAPR<n>**

GICC_NSAPR<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x00E0 + 4n</td>
<td>GICC_NSAPR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICC_PMR, CPU Interface Priority Mask Register

The GICC_PMR characteristics are:

Purpose

This register provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Note

Higher interrupt priority corresponds to a lower value of the Priority field.

Configuration

RW fields in this register reset to architecturally UNKNOWN values.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICC_PMR is a 32-bit register.

Field descriptions

The GICC_PMR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Priority |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the CPU interface. If the priority of the interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

If the GIC implementation supports fewer than 256 priority levels some bits might be RAZ/WI, as follows:

- For 128 supported levels, bit [0] = 0b0.
- For 64 supported levels, bits [1:0] = 0b00.
- For 32 supported levels, bits [2:0] = 0b000.
- For 16 supported levels, bits [3:0] = 0b0000.

See Interrupt prioritization for more information.

This field resets to an architecturally UNKNOWN value.

Accessing the GICC_PMR

If the GIC implementation supports two Security states:

- Non-secure accesses to this register can only read or write values corresponding to the lower half of the priority range.
If a Secure write has programmed the register with a value that corresponds to a value in the upper half of the priority range then:

- Any Non-secure read of the register returns 0x00, regardless of the value held in the register.
- Non-secure writes are ignored.

See 'Priority control of Secure and Non-secure interrupts' in the GICv3 Architecture Specification for more information.

**GICC_PMR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0004</td>
<td>GICC_PMR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICC_RPR, CPU Interface Running Priority Register

The GICC_RPR characteristics are:

**Purpose**

This register indicates the running priority of the CPU interface.

**Configuration**

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

**Attributes**

GICC_RPR is a 32-bit register.

**Field descriptions**

The GICC_RPR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Priority |

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR was set to the minimum value.

**Accessing the GICC_RPR**

If there is no active interrupt on the CPU interface, the idle priority value is returned.

If the GIC implementation supports two Security states, a Non-secure read of the Priority field returns:

- 0x00 if the field value is less than 0x80.
- The Non-secure view of the Priority value if the field value is 0x80 or more.

See 'Priority control of Secure and Non-secure interrupts' in the GICv3 Architecture Specification for more information.

**Note**

Software cannot determine the number of implemented priority bits from this register.
GICC_RPR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x0014</td>
<td>GICC_RPR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GICC_STATUSR, CPU Interface Status Register

The GICC_STATUSR characteristics are:

**Purpose**

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

**Configuration**

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

This register is used only when System register access is not enabled. If System register access is enabled, this register is not updated. Equivalent functionality might be provided by appropriate traps and exceptions.

**Attributes**

GICC_STATUSR is a 32-bit register.

**Field descriptions**

The GICC_STATUSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-5</td>
<td>Reserved</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>4</td>
<td>ASV</td>
<td>Attempted security violation.</td>
</tr>
<tr>
<td>3</td>
<td>WROD</td>
<td>Write to an RO location.</td>
</tr>
<tr>
<td>2</td>
<td>RWOD</td>
<td>Read of write-only location.</td>
</tr>
<tr>
<td>1</td>
<td>RWDRD</td>
<td>Write of read-only location.</td>
</tr>
</tbody>
</table>

**Bits [31:5]**

Reserved, RES0.

**ASV, bit [4]**

Attempted security violation.

<table>
<thead>
<tr>
<th>ASV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Non-secure access to a Secure register has been detected.</td>
</tr>
</tbody>
</table>

**Note**

This bit is not set to 1 for registers where any of the fields are Non-secure.

**WROD, bit [3]**

Write to an RO location.

<table>
<thead>
<tr>
<th>WROD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write to an RO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.
**RWOD, bit [2]**

Read of a WO location.

<table>
<thead>
<tr>
<th>RWOD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a WO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**WRD, bit [1]**

Write to a reserved location.

<table>
<thead>
<tr>
<th>WRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write to a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**RRD, bit [0]**

Read of a reserved location.

<table>
<thead>
<tr>
<th>RRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**Accessing the GICC_STATUSR**

This is an optional register. If the register is not implemented, the location is RAZ/WI.

If this register is implemented, GICV_STATUSR must also be implemented.

**GICC_STATUSR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x002C</td>
<td>GICC_STATUSR</td>
</tr>
<tr>
<td>(S)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC CPU interface</td>
<td>0x002C</td>
<td>GICC_STATUSR</td>
</tr>
<tr>
<td>(NS)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICD_CLRSPI_NSR characteristics are:

**Purpose**

Removes the pending state from a valid SPI if permitted by the Security state of the access and the \texttt{GICD_NSACR<n>} value for that SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

**Configuration**

If \texttt{GICD_TYPER.MBIS == 0}, this register is reserved.

When \texttt{GICD_CTLR.DS==1}, this register provides functionality for all SPIs.

**Attributes**

GICD_CLRSPI_NSR is a 32-bit register.

**Field descriptions**

The GICD_CLRSPI_NSR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>INTID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:13]**

Reserved, \texttt{RES0}.

**INTID, bits [12:0]**

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to \texttt{GICD_SETSPI_NSR} or \texttt{GICD_SETSPI_SR} adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to GICD_CLRSPI_NSR, \texttt{GICD_CLRSPI_SR}, or \texttt{GICD_ICPENDR<n>}.
- For a level-sensitive interrupt, a write to \texttt{GICD_SETSPI_NSR} or \texttt{GICD_SETSPI_SR} adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to GICD_CLRSPI_NSR or \texttt{GICD_CLRSPI_SR}. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

**Accessing the GICD_CLRSPI_NSR**

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding \texttt{GICD_NSACR<n>} register is less than \texttt{0b10}.
- The value written specifies an invalid SPI.
- The SPI is not pending.
16-bit accesses to bits [15:0] of this register must be supported.

**Note**

A Secure access to this register can clear the pending state of any valid SPI.

**GICD_CLRSPI_NSR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0048</td>
<td>GICD_CLRSPI_NSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICD_CLRSPI_SR, Clear Secure SPI Pending Register

The GICD_CLRSPI_SR characteristics are:

**Purpose**

Removes the pending state from a valid SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

**Configuration**

If `GICD_TYPER.MBIS == 0`, this register is reserved.

When `GICD_CTLR.DS==1`, this register is WI.

**Attributes**

GICD_CLRSPI_SR is a 32-bit register.

**Field descriptions**

The GICD_CLRSPI_SR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | INTID |

**Bits [31:13]**

Reserved, RES0.

**INTID, bits [12:0]**

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to `GICD_SETSPI_NSR` or `GICD_SETSPI_SR` adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to `GICD_CLRSPAN_SR` or `GICD_ICPENDR<n>`, then the interrupt will be active and pending.
- For a level-sensitive interrupt, a write to `GICD_SETSPI_NSR` or `GICD_SETSPI_SR` adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to `GICD_CLRSPAN_NSR` or `GICD_CLRSPAN_SR`. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

**Accessing the GICD_CLRSPI_SR**

 Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.
GICD_CLRSPI_SR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0058</td>
<td>GICD_CLRSPI_SR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WI**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WI**.
GICD_CPENDSGIR\langle n \rangle, SGI Clear-Pending Registers, n = 0 - 3

The GICD_CPENDSGIR\langle n \rangle characteristics are:

**Purpose**

Removes the pending state from an SGI.

A write to this register changes the state of a pending SGI to inactive, and the state of an active and pending SGI to active.

**Configuration**

Some or all RW fields of this register have defined reset values.

Four SGI clear-pending registers are implemented. Each register contains eight clear-pending bits for each of four SGIs, for a total of 16 possible SGIs.

In multiprocessor implementations, each PE has a copy of these registers.

**Attributes**

GICD_CPENDSGIR\langle n \rangle is a 32-bit register.

**Field descriptions**

The GICD_CPENDSGIR\langle n \rangle bit assignments are:

\[
\begin{array}{cccccccccccccccccccccccccc}
\hline
\text{SGI-clear\_pending\_bits<\langle x \rangle>, bits [8\langle x \rangle+7:8\langle x \rangle], for x = 0 to 3} \\
\end{array}
\]

**SGI_clear_pending_bits<\langle x \rangle>, bits [8\langle x \rangle+7:8\langle x \rangle], for x = 0 to 3**

Removes the pending state from SGI number 4n + x for the PE corresponding to the bit number written to.

Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>SGI_clear_pending_bits&lt;\langle x \rangle&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>If read, indicates that the SGI from the corresponding PE is not pending and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0x01</td>
<td>If read, indicates that the SGI from the corresponding PE is pending or is active and pending. If written, removes the pending state from the SGI for the corresponding PE.</td>
</tr>
</tbody>
</table>

This field resets to \(0\).

For SGI ID \(m\), generated by processing element \(C\) writing to the corresponding GICD_SGIR field, where DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_CPENDSGIR\langle n \rangle number is given by \(n = m \text{ DIV } 4\).
- The offset of the required register is \((0xF10 + (4n))\).
- The offset of the required field within the register GICD_CPENDSGIR\langle n \rangle is given by \(m \text{ MOD } 4\).
- The required bit in the 8-bit SGI clear-pending field \(m\) is bit \(C\).
Accessing the GICD_CPENDSGIR<n>

These registers are used only when affinity routing is not enabled. When affinity routing is enabled, this register is RES0. An implementation is permitted to make the register RAZ/WI in this case.

A register bit that corresponds to an unimplemented SGI is RAZ/WI.

These registers are byte-accessible.

If the GIC implementation supports two Security states:

- A register bit that corresponds to a Group 0 interrupt is RAZ/WI to Non-secure accesses.
- Register bits corresponding to unimplemented PEs are RAZ/WI.

**GICD_CPENDSGIR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0xF10 + 4n</td>
<td>GICD_CPENDSGIR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_CTLR, Distributor Control Register

The GICD_CTLR characteristics are:

**Purpose**

Enables interrupts and affinity routing.

**Configuration**

Some or all RW fields of this register have defined reset values.

The format of this register depends on the Security state of the access and the number of Security states supported, which is specified by GICD_CTLR.DS.

**Attributes**

GICD_CTLR is a 32-bit register.

**Field descriptions**

The GICD_CTLR bit assignments are:

**When access is Secure, in a system that supports two Security states:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RWP</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>7-4</td>
<td>E1NWF, DS, ARE_NS, ARE_S, RES0, EnableGrp1S, EnableGrp1NS, EnableGrp2NS</td>
</tr>
</tbody>
</table>

**RWP, bit [31]**

Register Write Pending. Read only. Indicates whether a register write is in progress or not:

<table>
<thead>
<tr>
<th>RWP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No register write in progress. The effects of previous register writes to the affected register fields are visible to all logical components of the GIC architecture, including the CPU interfaces.</td>
</tr>
<tr>
<td>1</td>
<td>Register write in progress. The effects of previous register writes to the affected register fields are not guaranteed to be visible to all logical components of the GIC architecture, including the CPU interfaces, as the effects of the changes are still being propagated.</td>
</tr>
</tbody>
</table>

This field tracks writes to:

- GICD_CTLR[2:0], the Group Enables, for transitions from 1 to 0 only.
- GICD_CTLR[7:4], the ARE bits, E1NWF bit and DS bit.
- GICD_ICENABLER<n>.

Updates to other register fields are not tracked by this field.

This field resets to an architecturally UNKNOWN value.

**Bits [30:8]**

Reserved, RES0.
**E1NWF, bit [7]**

Enable 1 of N Wakeup Functionality.

It is **IMPLEMENTATION DEFINED** whether this bit is programmable, or RAZ/WI.

If it is implemented, then it has the following behavior:

<table>
<thead>
<tr>
<th>E1NWF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A PE that is asleep cannot be picked for 1 of N interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>A PE that is asleep can be picked for 1 of N interrupts as determined by <strong>IMPLEMENTATION DEFINED</strong> controls.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**DS, bit [6]**

Disable Security.

<table>
<thead>
<tr>
<th>DS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure accesses are not permitted to access and modify registers that control Group 0 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure accesses are permitted to access and modify registers that control Group 0 interrupts.</td>
</tr>
</tbody>
</table>

If DS is written from 0 to 1 when GICD_CTLR.ARE_S == 1, then GICD_CTLR.ARE for the single Security state is RAO/WI.

If the Distributor only supports a single Security state, this bit is RAO/WI.

If the Distributor supports two Security states, it **IMPLEMENTATION DEFINED** whether this bit is programmable or implemented as RAZ/WI.

When this field is set to 1, all accesses to GICD_CTLR access the single Security state view, and all bits are accessible.

When set to 1, this field can only be cleared by a hardware reset.

Writing this bit from 0 to 1 is **UNPREDICTABLE** if any of the following is true:

- GICD_CTLR.EnableGrp0 == 1.
- GICD_CTLR.EnableGrp1S == 1.
- GICD_CTLR.EnableGrp1NS == 1.
- One or more INTID is in the Active or Active and Pending state.

This field resets to 0.

**ARE_NS, bit [5]**

Affinity Routing Enable, Non-secure state.

<table>
<thead>
<tr>
<th>ARE_NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Affinity routing disabled for Non-secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Affinity routing enabled for Non-secure state.</td>
</tr>
</tbody>
</table>

When affinity routing is enabled for the Secure state, this field is RAO/WI.

Changing the ARE_NS settings from 0 to 1 is **UNPREDICTABLE** except when GICD_CTLR.EnableGrp1 Non-secure == 0.

Changing the ARE_NS settings from 1 to 0 is **UNPREDICTABLE**.

If GICv2 backwards compatibility for Non-secure state is not implemented, this field is RAO/WI.

This field resets to 0.

**ARE_S, bit [4]**

Affinity Routing Enable, Secure state.
Changing the ARE_S setting from 0 to 1 is UNPREDICTABLE except when all of the following apply:

- GICD_CTLR.EnableGrp0==0.
- GICD_CTLR.EnableGrp1S==0.
- GICD_CTLR.EnableGrp1NS==0.

Changing the ARE_S settings from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility for Secure state is not implemented, this field is RAO/WI.

This field resets to 0.

**Bit [3]**

Reserved, RES0.

**EnableGrp1S, bit [2]**

Enable Secure Group 1 interrupts.

<table>
<thead>
<tr>
<th>EnableGrp1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

If GICD_CTLR.ARE_S == 0, this field is RES0.

This field resets to an architecturally UNKNOWN value.

**EnableGrp1NS, bit [1]**

Enable Non-secure Group 1 interrupts.

<table>
<thead>
<tr>
<th>EnableGrp1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure Group 1 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure Group 1 interrupts are enabled.</td>
</tr>
</tbody>
</table>

**Note**

This field also controls whether LPIs are forwarded to the PE.

This field resets to an architecturally UNKNOWN value.

**EnableGrp0, bit [0]**

Enable Group 0 interrupts.

<table>
<thead>
<tr>
<th>EnableGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**When access is Non-secure, in a system that supports two Security states:**

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWP  RES0  ARE_NS  RES0 EnableGrp1A  EnableGrp1</td>
</tr>
</tbody>
</table>
RWP, bit [31]

This bit is a read-only alias of the Secure GICD_CTLR.RWP bit.

Bits [30:5]

Reserved, RES0.

ARE_NS, bit [4]

This bit is a read-write alias of the Secure GICD_CTLR.ARE_NS bit.

If GICv2 backwards compatibility for Non-secure state is not implemented, this field is RAO/WI.

Bits [3:2]

Reserved, RES0.

EnableGrp1A, bit [1]

If ARE_NS == 1, then this bit is a read-write alias of the Secure GICD_CTLR.EnableGrp1NS bit.

If ARE_NS == 0, then this bit is RES0.

EnableGrp1, bit [0]

If ARE_NS == 0, then this bit is a read-write alias of the Secure GICD_CTLR.EnableGrp1NS bit.

If ARE_NS == 1, then this bit is RES0.

When in a system that supports only a single Security state:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No register write in progress. The effects of previous register writes to the affected register fields are visible to all logical components of the GIC architecture, including the CPU interfaces.</td>
</tr>
<tr>
<td>0b1</td>
<td>Register write in progress. The effects of previous register writes to the affected register fields are not guaranteed to be visible to all logical components of the GIC architecture, including the CPU interfaces, as the effects of the changes are still being propagated.</td>
</tr>
</tbody>
</table>

This field tracks updates to:

- GICD_CTLR[2:0], the Group Enables, for transitions from 1 to 0 only.
- GICD_CTLR[7:4], the ARE bits, E1NWF bit and DS bit.
- GICD_ICENABLER<n>, the bits that allow disabling of SPIs.

Updates to other register fields are not tracked by this field.

This field resets to an architecturally UNKNOWN value.
Bits [30:8]

Reserved, RES0.

E1NWF, bit [7]

Enable 1 of N Wakeup Functionality.

It is IMPLEMENTATION DEFINED whether this bit is programmable, or RAZ/WI.

If it is implemented, then it has the following behavior:

<table>
<thead>
<tr>
<th>E1NWF</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A PE that is asleep cannot be picked for 1 of N interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>A PE that is asleep can be picked for 1 of N interrupts as determined by IMPLEMENTATION DEFINED controls.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

DS, bit [6]

Disable Security. This field is RAO/WI.

Bit [5]

Reserved, RES0.

ARE, bit [4]

Affinity Routing Enable.

<table>
<thead>
<tr>
<th>ARE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Affinity routing disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Affinity routing enabled.</td>
</tr>
</tbody>
</table>

Changing the ARE settings from 0 to 1 is UNPREDICTABLE except when all of the following apply:

- GICD_CTLR.EnableGrp1==0.
- GICD_CTLR.EnableGrp0==0.

Changing ARE from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility is not implemented, this field is RAO/WI.

This field resets to 0.

Bits [3:2]

Reserved, RES0.

EnableGrp1, bit [1]

Enable Group 1 interrupts.

<table>
<thead>
<tr>
<th>EnableGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 interrupts disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 interrupts enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

Enable Group 0 interrupts.
<table>
<thead>
<tr>
<th>EnableGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 interrupts are enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

## Accessing the GICD_CTLR

If an interrupt is pending within a CPU interface when the corresponding GICD_CTLR.EnableGrpX bit is written from 1 to 0 the interrupt must be retrieved from the CPU interface.

### Note

This might have no effect on the forwarded interrupt if it has already been activated. When a write changes the value of ARE for a Security state or the value of the DS bit, the format used for interpreting the remaining bits provided in the write data is the format that applied before the write takes effect.

---

**GICD_CTLR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0000</td>
<td>GICD_CTLR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_ICACTIVER<n>, Interrupt Clear-Active Registers, n = 0 - 31

The GICD_ICACTIVER<n> characteristics are:

**Purpose**

Deactivates the corresponding interrupt. These registers are used when saving and restoring GIC state.

**Configuration**

Some or all RW fields of this register have defined reset values.

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ICACTIVER<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ICACTIVER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ICACTIVER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:
  - Register is RAZ/WI.
  - An UNKNOWN banked copy of the register is accessed.

**Attributes**

GICD_ICACTIVER<n> is a 32-bit register.

**Field descriptions**

The GICD_ICACTIVER<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Clear_active_bit<x>, bit [x], for x = 0 to 31 |

**Clear_active_bit<x>, bit [x], for x = 0 to 31**

Removes the active state from interrupt number 32n + x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_active_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:
  - The corresponding GICD_ICACTIVER<n> number, n, is given by n = m DIV 32.
  - The offset of the required GICD_ICACTIVER is (0x380 + (4*n)).
  - The bit number of the required group modifier bit in this register is m MOD 32.
Accessing the GICD_ICACTIVER<n>

When affinity routing is enabled for the Security state of an interrupt, the bits corresponding to SGIs and PPIs in that Security state are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ICACTIVER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

GICD_ICACTIVER<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0380 + 4n</td>
<td>GICD_ICACTIVER&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICD_ICACTIVER<n>E characteristics are:

**Purpose**

Removes the active state from the corresponding SPI in the extended SPI range.

**Configuration**

Some or all RW fields of this register have defined reset values.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICACTIVER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

**Attributes**

GICD_ICACTIVER<n>E is a 32-bit register.

**Field descriptions**

The GICD_ICACTIVER<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Clear_active_bit&lt;x&gt;, bit [x], for x = 0 to 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

For the extended SPIs, removes the active state to interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_active_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICACTIVER<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ICACTIVER<n>E is (0x1C00 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

**Accessing the GICD_ICACTIVER<n>E**

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICACTIVER<n>E, the corresponding bit is RES0.
When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICD_ICACTIVER<n>E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x1C00 + 4n</td>
<td>GICD_ICACTIVER&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICD_ICENABLER\(<n>\) characteristics are:

**Purpose**

Disables forwarding of the corresponding interrupt to the CPU interfaces.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ICENABLER\(<n>\) registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ICENABLER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ICENABLER0 from a PE with GICR_TYPER.Processor_Number > 7 is **CONSTRAINED UNPREDICTABLE**:

- Register is RAZ/WI.
- An **UNKNOWN** banked copy of the register is accessed.

**Attributes**

GICD_ICENABLER\(<n>\) is a 32-bit register.

**Field descriptions**

The GICD_ICENABLER\(<n>\) bit assignments are:

| Clear_enable_bit\(<x>\), bit \([x]\), for \(x = 0 \text{ to } 31\) |
|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Clear_enable_bit\(<x>\), bit \([x]\), for \(x = 0 \text{ to } 31\) |

For SPIs and PPIs, controls the forwarding of interrupt number \(32n + x\) to the CPU interfaces. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_enable_bit(&lt;x&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0b0)</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>(0b1)</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.</td>
</tr>
</tbody>
</table>

For SGIs, the behavior of this bit is **IMPLEMENTATION DEFINED**.

This field resets to an architecturally **UNKNOWN** value.

For INTID \(m\), when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICENABLER\(<n>\) number, \(n\), is given by \(n = m \text{ DIV } 32\).
- The offset of the required GICD_ICENABLER is \((0x180 + (4*n))\).
The bit number of the required group modifier bit in this register is \( m \mod 32 \).

**Note**

Writing a 1 to a GICD_ICENABLER\(<n>\) bit only disables the forwarding of the corresponding interrupt from the Distributor to any CPU interface. It does not prevent the interrupt from changing state, for example becoming pending or active and pending if it is already active.

### Accessing the GICD_ICENABLER\(<n>\)

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is \( \text{RES0} \) and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ICENABLER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When GICD_CTLR.DS==0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

It is **IMPLEMENTATION DEFINED** whether implemented SGIs are permanently enabled, or can be enabled and disabled by writes to GICD_ISENABLER\(<n>\) and GICD_ICENABLER\(<n>\) where \( n=0 \).

Completion of a write to this register does not guarantee that the effects of the write are visible throughout the affinity hierarchy. To ensure an enable has been cleared, software must write to the register with bits set to 1 to clear the required enables. Software must then poll GICD_CTLR.RWP until it has the value zero.

GICD_ICENABLER\(<n>\) can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0180 + 4n</td>
<td>GICD_ICENABLER(&lt;n&gt;)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_ICENABLER<n>E, Interrupt Clear-Enable Registers, n = 0 - 31

The GICD_ICENABLER<n>E characteristics are:

**Purpose**

Disables forwarding of the corresponding SPI in the extended SPI range to the CPU interfaces.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICD_ICENABLER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICENABLER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

**Attributes**

GICD_ICENABLER<n>E is a 32-bit register.

**Field descriptions**

The GICD_ICENABLER<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 - 0</td>
<td>Clear_enable_bit&lt;x&gt;, bit [x], for x = 0 to 31</td>
</tr>
</tbody>
</table>

For the extended SPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_enable_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICENABLER<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_ICENABLER<n>E is (0x1400 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.
Accessing the GICD_ICENABLER\textlt n\textgt E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICENABLER\textlt n\textgt E, the corresponding bit is RES0.

When \texttt{GICD_CTLR.DS} = 0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

\textbf{GICD_ICENABLER\textlt n\textgt E can be accessed through the memory-mapped interfaces:}

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x1400  + 4n</td>
<td>GICD_ICENABLER\textlt n\textgt E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{GICD_CTLR.DS} == 0b0 accesses to this register are \textbf{RW}.
- When \texttt{IsAccessSecure()} accesses to this register are \textbf{RW}.
- When \texttt{!IsAccessSecure()} accesses to this register are \textbf{RW}.  

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**GICD_ICFGR<n>, Interrupt Configuration Registers, n = 0 - 63**

The GICD_ICFGR<n> characteristics are:

**Purpose**

Determines whether the corresponding interrupt is edge-triggered or level-sensitive.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

These registers are available in all GIC configurations. If the GIC implementation supports two Security states, these registers are **Common**.

GICD_ICFGR1 is Banked for each connected PE with **GICR_TYPER**.Processor_Number < 8.

Accessing GICD_ICFGR1 from a PE with **GICR_TYPER**.Processor_Number > 7 is **CONSTRAINED UNPREDICTABLE**:

- Register is RAZ/WI.
- An **UNKNOWN** banked copy of the register is accessed.

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is **RES0** and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ICFGR<n>

For each supported PPI, it is **IMPLEMENTATION DEFINED** whether software can program the corresponding Int_config field.

For SGIs, Int_config fields are RO, meaning that GICD_ICFGR0 is RO.

Changing Int_config when the interrupt is individually enabled is **UNPREDICTABLE**.

Changing the interrupt configuration between level-sensitive and edge-triggered (in either direction) at a time when there is a pending interrupt will leave the interrupt in an **UNKNOWN** pending state.

Fields corresponding to unimplemented interrupts are RAZ/WI.

**Attributes**

GICD_ICFGR<n> is a 32-bit register.

**Field descriptions**

The GICD_ICFGR<n> bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Int_config<x>, bits [2x+1:2x], for x = 0 to 15**

Indicates whether the interrupt with ID 16n + x is level-sensitive or edge-triggered.

Int_config[0] (bit [2x]) is **RES0**.

Possible values of Int_config[1] (bit [2x+1]) are:
### Int_config<x>

<table>
<thead>
<tr>
<th>Int_config&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Corresponding interrupt is level-sensitive.</td>
</tr>
<tr>
<td>0b01</td>
<td>Corresponding interrupt is edge-triggered.</td>
</tr>
</tbody>
</table>

For SGIs, Int_config[1] is RAO/WI.

For SPIs and PPIs, Int_config[1] is programmable unless the implementation supports two Security states and the bit corresponds to a Group 0 or Secure Group 1 interrupt, in which case the bit is RAZ/WI to Non-secure accesses.

This field resets to an architecturally **UNKNOWN** value.

## Accessing the GICD_ICFGR<n>

**GICD_ICFGR<n>** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0xC00 + 4n</td>
<td>GICD_ICFGR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_ICFGR\(<n>E\), Interrupt Configuration Registers (Extended SPI Range), \(n = 0 - 63\)

The GICD_ICFGR\(<n>E\) characteristics are:

**Purpose**

Determines whether the corresponding SPI in the extended SPI range is edge-triggered or level-sensitive.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when GIC, \(\geq 3.1\) is implemented. Otherwise, direct accesses to GICD_ICFGR\(<n>E\) are **RES0**.

When GICD_TYPER.ESPI==0, these registers are **RES0**.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICFGR\(<n>E\) registers is \(((GICD_TYPER.ESPI\_range+1)*2)\). Registers are numbered from 0.

**Attributes**

GICD_ICFGR\(<n>E\) is a 32-bit register.

**Field descriptions**

The GICD_ICFGR\(<n>E\) bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **Int_config\(<x>, bits \([2x+1:2x]\), for \(x = 0\) to 15** |

**Int_config\(<x>, bits \([2x+1:2x]\), for \(x = 0\) to 15**

Indicates whether the interrupt with ID \(16n + x\) is level-sensitive or edge-triggered.

Int_config[0] (bit[2x]) is **RES0**.

Possible values of Int_config[1] (bit[2x+1]) are:

<table>
<thead>
<tr>
<th>Int_config(&lt;x&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Corresponding interrupt is level-sensitive.</td>
</tr>
<tr>
<td>0b01</td>
<td>Corresponding interrupt is edge-triggered.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Accessing the GICD_ICFGR\(<n>E\)**

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICFGR\(<n>E\), the corresponding bit is **RES0**.

When **GICD_CTLR.DS==0**, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.
GICD_ICFGR\<n\>E can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0×3000 + 4(n)</td>
<td>GICD_ICFGR&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_ICPENDR<n>, Interrupt Clear-Pending Registers, n = 0 - 31

The GICD_ICPENDR<n> characteristics are:

**Purpose**

Removes the pending state from the corresponding interrupt.

**Configuration**

Some or all RW fields of this register have defined reset values.

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ICPENDR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ICPENDR0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ICPENDR0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

**Attributes**

GICD_ICPENDR<n> is a 32-bit register.

**Field descriptions**

The GICD_ICPENDR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Clear_pending_bit<x>, bit [x], for x = 0 to 31 |

**Clear_pending_bit<x>, bit [x], for x = 0 to 31**

For SPIs and PPIs, removes the pending state from interrupt number 32n + x. Reads and writes have the following behavior:
### Clear_pending_bit<x>

<table>
<thead>
<tr>
<th>Clear_pending_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending on any PE. If written, has no effect.</td>
</tr>
</tbody>
</table>
| 0b1                  | If read, indicates that the corresponding interrupt is pending, or active and pending:  
  - On this PE if the interrupt is an SGI or PPI.  
  - On at least one PE if the interrupt is an SPI.  
If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases:  
  - If the interrupt is an SGI. In this case, the write is ignored. The pending state of an SGI can be cleared using GICD_CPENDSGIR<n>.  
  - If the interrupt is not pending and is not active and pending.  
  - If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.  
This field resets to 0. |

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICPENDR<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ICPENDR is (0x200 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

### Accessing the GICD_ICPENDR<n>

Clear-pending bits for SGIs are RO/WI.

When affinity routing is enabled for the Security state of an interrupt:

- Bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ICPENDR0.
- Bits corresponding to Group 0 and Group 1 Secure interrupts can only be cleared by Secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

**GICD_ICPENDR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0280 + 4n</td>
<td>GICD_ICPENDR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The GICD_ICPENDR\(n\)E characteristics are:

**Purpose**

Removes the pending state to the corresponding SPI in the extended SPI range.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, \(>=3.1\) is implemented. Otherwise, direct accesses to GICD_ICPENDR\(n\)E are \(\text{RES}\)0.

When \(\text{GICD_TYPER}.\text{ESPI}==0\), these registers are \(\text{RES}\)0.

When \(\text{GICD_TYPER}.\text{ESPI}==1\), the number of implemented GICD_ICPENDR\(n\)E registers is \((\text{GICD_TYPER}.\text{ESPI}_\text{range}+1)\). Registers are numbered from 0.

**Attributes**

GICD_ICPENDR\(n\)E is a 32-bit register.

**Field descriptions**

The GICD_ICPENDR\(n\)E bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignments</th>
<th>Clear_pending_bit(x), bit [x], for (x = 0) to (31)</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
</tbody>
</table>

For the extended PPIs, removes the pending state to interrupt number \(x\). Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_pending_bit(x)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is not pending and is not active and pending.</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR(n)E. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.</td>
</tr>
</tbody>
</table>

This field resets to 0.
For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICPENDR<n>E number, n, is given by \( n = (m - 4096) \div 32 \).
- The offset of the required GICD_ICPENDR<n>E is \( (0x1800 + (4n)) \).
- The bit number of the required group modifier bit in this register is \( (m - 4096) \mod 32 \).

**Accessing the GICD_ICPENDR<n>E**

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICPENDR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS == 0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICD_ICPENDR<n>E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x1800 + 4n</td>
<td>GICD_ICPENDR&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICD_IGROUPR<n> characteristics are:

**Purpose**

Controls whether the corresponding interrupt is in Group 0 or Group 1.

**Configuration**

GICD_IGROUPR0 resets to an IMPLEMENTATION DEFINED value, that might be UNKNOWN.

GICD_IGROUPR<n> where n is greater than 0 resets to 0x00000000.

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Secure.

The number of implemented GICD_IGROUPR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_IGROUPR0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_IGROUPR0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

**Attributes**

GICD_IGROUPR<n> is a 32-bit register.

**Field descriptions**

The GICD_IGROUPR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Group_status_bit<x>, bit [x], for x = 0 to 31

**Group_status_bit<x>, bit [x], for x = 0 to 31**

Group status bit.

<table>
<thead>
<tr>
<th>Group_status_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When GICD_CTLR.DS==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS==0, the corresponding interrupt is Secure.</td>
</tr>
<tr>
<td>0b1</td>
<td>When GICD_CTLR.DS==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.</td>
</tr>
</tbody>
</table>

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGRPMODR<n> to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICD_IGRPMODR<n>.

If affinity routing is disabled for the Security state of an interrupt, then:

- The corresponding GICD_IGRPMODR<n> bit is RES0.
- For Secure interrupts, the interrupt is Secure Group 0.
For Non-secure interrupts, the interrupt is Non-secure Group 1.

For INTID m, when DIV and MOD are the integer division and modulo operations:

• The corresponding GICD_IGROUP<n> number, n, is given by \( n = m \div 32 \).
• The offset of the required GICD_IGROUP is \((0x080 + (4^n))\).
• The bit number of the required group modifier bit in this register is \( m \mod 32 \).

## Accessing the GICD_IGROUPR<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_IGROUPR0.

When \( \text{GICD_CTLR.DS} = 0 \), the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

### Note

Accesses to GICD_IGROUPR0 when affinity routing is not enabled for a Security state access the same state as \( \text{GICR_IGROUPR0} \), and must update Redistributor state associated with the PE performing the accesses. Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

### GICD_IGROUPR<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0080 + 4n</td>
<td>GICD_IGROUPR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
GICD_IGROUPR<n>E, Interrupt Group Registers (extended SPI range), n = 0 - 31

The GICD_IGROUPR<n>E characteristics are:

**Purpose**

Controls whether the corresponding SPI in the extended SPI range is in Group 0 or Group 1.

**Configuration**

GICD_IGROUPR<n>E resets to 0x00000000.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1:

- The number of implemented GICD_IGROUPR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.
- When GICD_CLTR.DS==0, this register is Secure.

**Attributes**

GICD_IGROUPR<n>E is a 32-bit register.

**Field descriptions**

The GICD_IGROUPR<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Group_status_bit&lt;x&gt;, bit [x], for x = 0 to 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**Group_status_bit<x>, bit [x], for x = 0 to 31**

Group status bit.

<table>
<thead>
<tr>
<th>Group_status_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When GICD_CTLR.DS==1, the corresponding interrupt is Group 0.</td>
</tr>
<tr>
<td></td>
<td>When GICD_CTLR.DS==0, the corresponding interrupt is Secure.</td>
</tr>
<tr>
<td>0b1</td>
<td>When GICD_CTLR.DS==1, the corresponding interrupt is Group 1.</td>
</tr>
<tr>
<td></td>
<td>When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.</td>
</tr>
</tbody>
</table>

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGRPMODR<n>E to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICD_IGRPMODR<n>E.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGROUPR<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_IGROUPR<n>E is (0x1000 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.
Accessing the GICD_IGROUPR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IGROUPR<n>E can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x1000 + 4n</td>
<td>GICD_IGROUPR&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICD_IGRPMODR<n> characteristics are:

**Purpose**

When GICD_CTLR.DS==0, this register together with the GICD_IGROUPR<n> registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- Secure Group 1.

**Configuration**

Some or all RW fields of this register have defined reset values.

When GICD_CLTR.DS==0, these registers are Secure.

The number of implemented GICD_IGROUPR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

When GICD_CTLR.ARE_S==0 or GICD_CTLR.DS==1, the GICD_IGRPMODR<n> registers are RES0. An implementation can make these registers RAZ/WI in this case.

**Attributes**

GICD_IGRPMODR<n> is a 32-bit register.

**Field descriptions**

The GICD_IGRPMODR<n> bit assignments are:

```
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| Group_modifier_bit<x>, bit [x], for x = 0 to 31 |
```

**Group_modifier_bit<x>, bit [x], for x = 0 to 31**

Group modifier bit. When affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICD_IGROUPR<n> to form a 2-bit field that defines an interrupt group:

<table>
<thead>
<tr>
<th>Group modifier bit</th>
<th>Group status bit</th>
<th>Definition</th>
<th>Short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b0</td>
<td>Secure Group 0</td>
<td>G0S</td>
</tr>
<tr>
<td>0b0</td>
<td>0b1</td>
<td>Non-secure Group 1</td>
<td>G1NS</td>
</tr>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>Secure Group 1</td>
<td>G1S</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>Reserved, treated as Non-secure Group 1</td>
<td>-</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGRPMODR<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_IGRPMODR is (0x080 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

See GICD_IGROUPR<n> for information about the GICD_IGRPMODR0 reset value.
Accessing the GICD_IGRPMODR<n>

When affinity routing is enabled for Secure state, GICD_IGRPMODR0 is RES0 and equivalent functionality is proved by GICR_IGRPMODR0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

---

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

---

GICD_IGRPMODR<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0D00 + 4n</td>
<td>GICD_IGRPMODR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
GICD_IGRPMODR<n>E, Interrupt Group Modifier Registers (extended SPI range), n = 0 - 31

The GICD_IGRPMODR<n>E characteristics are:

**Purpose**

When GICD_CTLR.DS==0, this register together with the GICD_IGROUPR<n>E registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

**Configuration**

Some or all RW fields of this register have defined reset values.

GICD_IGRPMODR<n>E resets to 0x00000000.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1:

- The number of implemented GICD_IGRPMODR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.
- When GICD_CLTR.DS==0, this register is Secure.

**Attributes**

GICD_IGRPMODR<n>E is a 32-bit register.

**Field descriptions**

The GICD_IGRPMODR<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Bit position</th>
<th>Group modifier bit</th>
<th>Group status bit</th>
<th>Definition</th>
<th>Short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 - 24</td>
<td>0b0</td>
<td>0b0</td>
<td>Secure Group 0</td>
<td>G0S</td>
</tr>
<tr>
<td>24 - 16</td>
<td>0b0</td>
<td>0b1</td>
<td>Non-secure Group 1</td>
<td>G1NS</td>
</tr>
<tr>
<td>16 - 8</td>
<td>0b1</td>
<td>0b0</td>
<td>Secure Group 1</td>
<td>G1S</td>
</tr>
<tr>
<td>8 - 0</td>
<td>0b1</td>
<td>0b1</td>
<td>Reserved, treated as Non-secure Group 1</td>
<td>-</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGRPMODR<n>E number, n, is given by n = (m-4096) DIV 32.
- The offset of the required GICD_IGRPMODR<n>E is (0x3400 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.
Accessing the GICD\_IGRPMODR\(<n>E\)

When affinity routing is not enabled for the Security state of an interrupt in GICD\_IGRPMODR\(<n>E\), the corresponding bit is RES0.

When GICD\_CTL\_RS == 0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD\_IGRPMODR\(<n>E\) can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x3400 + 4n</td>
<td>GICD_IGRPMODR(&lt;n&gt;E)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD\_CTL\_DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICD_IIDR characteristics are:

**Purpose**

Provides information about the implementer and revision of the Distributor.

**Configuration**

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

**Attributes**

GICD_IIDR is a 32-bit register.

**Field descriptions**

The GICD_IIDR bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |

**ProductID**, bits [31:24]

An IMPLEMENTATION DEFINED product identifier.

**Bits [23:20]**

Reserved, RES0.

**Variant, bits [19:16]**

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

**Revision, bits [15:12]**

An IMPLEMENTATION DEFINED revision number. Typically, this field is used to distinguish minor revisions of a product.

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the Distributor:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x38.
Accessing the GICD_IIDR

GICD_IIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0008</td>
<td>GICD_IIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRL.RS == 0b0 accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GICD_IPRIORITYR<n>, Interrupt Priority Registers, n = 0 - 254

The GICD_IPRIORITYR<n> characteristics are:

**Purpose**

Holds the priority of the corresponding interrupt.

**Configuration**

Some or all RW fields of this register have defined reset values.

These registers are available in all configurations of the GIC. When GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_IPRIORITYR<n> registers is 8*(GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_IPRIORITYR0 to GICD_IPRIORITYR7 are Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_IPRIORITYR0 to GICD_IPRIORITYR7 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

**Attributes**

GICD_IPRIORITYR<n> is a 32-bit register.

**Field descriptions**

The GICD_IPRIORITYR<n> bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Priority_offset_3B | Priority_offset_2B | Priority_offset_1B | Priority_offset_0B |

**Priority_offset_3B, bits [31:24]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

This field resets to 0.

**Priority_offset_2B, bits [23:16]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

This field resets to 0.

**Priority_offset_1B, bits [15:8]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

This field resets to 0.
Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.

This field resets to 0.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IPRIORITYR<n> number, n, is given by n = m DIV 4.
- The offset of the required GICD_IPRIORITYR<n> register is (0x400 + (4*n)).
- The byte offset of the required Priority field in this register is m MOD 4, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Accessing the GICD_IPRIORITYR<n>

These registers are always used when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt:

- GICR_IPRIORITYR<n> is used instead of GICD_IPRIORITYR<n> where n = 0 to 7 (that is, for SGIs and PPIs).
- GICD_IPRIORITYR<n> is RAZ/WI where n = 0 to 7.

These registers are byte-accessible.

A register field corresponding to an unimplemented interrupt is RAZ/WI.

A GIC might implement fewer than eight priority bits, but must implement at least bits [7:4] of each field. In each field, unimplemented bits are RAZ/WI, see Interrupt prioritization.

When GICD_CTLR.DS==0:

- A register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software views of interrupt priority.

It is IMPLEMENTATION DEFINED whether changing the value of a priority field changes the priority of an active interrupt.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IPRIORITYR<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x400 + 4n</td>
<td>GICD_IPRIORITYR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GIC_DCTRL.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GICD_IPRIORITYR<n>E, Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC., n = 0 - 255

The GICD_IPRIORITYR<n>E characteristics are:

**Purpose**

Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICD_IPRIORITYR<n>E are RES0.

When **GICD_TYPER**.ESPI==0, these registers are RES0.

When **GICD_TYPER**.ESPI==1, the number of implemented GICD_IPRIORITYR<n>E registers is ((**GICD_TYPER**.ESPI_range+1)*8). Registers are numbered from 0.

**Attributes**

GICD_IPRIORITYR<n>E is a 32-bit register.

**Field descriptions**

The GICD_IPRIORITYR<n>E bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Priority_offset_3B | Priority_offset_2B | Priority_offset_1B | Priority_offset_0B |

**Priority_offset_3B**, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.

**Priority_offset_2B**, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.

**Priority_offset_1B**, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.
Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IPRIORITYRₙE number, n, is given by \( n = (m-4096) \div 4 \).
- The offset of the required GICD_IPRIORITYRₙE register is \( 0x2000 + (4*n) \).
- The byte offset of the required Priority field in this register is \( m \mod 4 \), where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Accessing the GICD_IPRIORITYRₙE

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISACTIVERₙE, the corresponding bit is RES0.

When \texttt{GICD_CTLR.DS==0}:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

Bits corresponding to unimplemented interrupts are RAZ/WI.

\begin{center}
\textbf{Note}
\end{center}

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than once. The effect of the change must be visible in finite time.

GICD_IPRIORITYRₙE can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x2000 + 4n</td>
<td>GICD_IPRIORITYRₙE</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{GICD_CTLR.DS == 0b0} accesses to this register are \texttt{RW}.
- When \texttt{IsAccessSecure()} accesses to this register are \texttt{RW}.
- When \texttt{!IsAccessSecure()} accesses to this register are \texttt{RW}.
The GICD_IROUTER<n> characteristics are:

**Purpose**

When affinity routing is enabled, provides routing information for the SPI with INTID n.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

These registers are available in all configurations of the GIC. If the GIC implementation supports two Security states, these registers are Common.

The maximum value of n is given by \((32*(GICD_TYPER.ITLinesNumber+1) - 1)\). GICD_IROUTER<n> registers where n=0 to 31 are reserved.

**Attributes**

GICD_IROUTER<n> is a 64-bit register.

**Field descriptions**

The GICD_IROUTER<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>62</td>
<td>61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32</td>
</tr>
<tr>
<td>31</td>
<td>Interrupt_Routing_Mode</td>
</tr>
<tr>
<td>30</td>
<td>29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**Bits [63:40]**

Reserved, RES0.

**Aff3, bits [39:32]**

Affinity level 3, the least significant affinity level field.

This field resets to an architecturally UNKNOWN value.

**Interrupt_Routing_Mode, bit [31]**

Interrupt Routing Mode. Defines how SPIs are routed in an affinity hierarchy:

<table>
<thead>
<tr>
<th>Interrupt_Routing_Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PE specified by a.b.c.d. In this routing, a, b, c, and d are the values of fields Aff3, Aff2, Aff1, and Aff0 respectively.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to any PE defined as a participating node.</td>
</tr>
</tbody>
</table>

If GICD_IROUTER<n>.IRM == 0 and the affinity path does not correspond to an implemented PE, then if the corresponding interrupt becomes pending it will not be forwarded to any PE and will remain pending.
In implementations that do not require 1 of N distribution of SPIs, this bit might be RAZ/WI.

When this bit is set to 1, GICD_IROUTER<n>.\{Aff3, Aff2, Aff1, Aff0\} are \textit{UNKNOWN}.

\textbf{Note}

An implementation might choose to make the Aff<n> fields RO when this field is 1.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{Bits [30:24]}

Reserved, RES0.

\textbf{Aff2, bits [23:16]}

Affinity level 2, an intermediate affinity level field.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{Aff1, bits [15:8]}

Affinity level 1, an intermediate affinity level field.

This field resets to an architecturally \textit{UNKNOWN} value.

\textbf{Aff0, bits [7:0]}

Affinity level 0, the most significant affinity level field.

This field resets to an architecturally \textit{UNKNOWN} value.

For an SPI with INTID m:

\begin{itemize}
  \item The corresponding GICD_IROUTER<n> register number, n, is given by n = m.
  \item The offset of the GICD_IROUTER<n> register is 0x6000 + 8n.
\end{itemize}

\textbf{Accessing the GICD_IROUTER<n>}

These registers are used only when affinity routing is enabled. When affinity routing is not enabled:

\begin{itemize}
  \item These registers are RES0. An implementation is permitted to make the register RAZ/WI in this case.
  \item The GICD_ITARGETSR<n> registers provide interrupt routing information.
\end{itemize}

\textbf{Note}

When affinity routing becomes enabled for a Security state (for example, following a reset or following a write to GICD_CTLR) the value of all writeable fields in this register is \textit{UNKNOWN} for that Security state. When the group of an interrupt changes so the ARE setting for the interrupt changes to 1, the value of this register is \textit{UNKNOWN} for that interrupt.

\textbf{Note}

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any GICD_IROUTER<n> registers that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

\textbf{Note}

For each interrupt, a GIC implementation might support fewer than 256 values for an affinity level. In this case, some bits of the corresponding affinity level field might be RO. Implementations must ensure that an interrupt that is
pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

**GICD_IROUTER<n>** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x6000 + 8n</td>
<td>GICD_IROUTER&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICD_IROUTER<n>E characteristics are:

**Purpose**

When affinity routing is enabled, provides routing information for the corresponding SPI in the extended SPI range.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

When `GICD_TYPER.ESPI==0`, these registers are **RES0**.

When `GICD_TYPER.ESPI==1`, the number of implemented GICD_IROUTER<n>E registers is `((GICD_TYPER.ESPI_range+1)*32)-1`). Registers are numbered from 0.

**Attributes**

GICD_IROUTER<n>E is a 32-bit register.

**Field descriptions**

The GICD_IROUTER<n>E bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RES0</strong></td>
<td><strong>Aff3</strong></td>
<td><strong>Aff2</strong></td>
</tr>
<tr>
<td><strong>Aff1</strong></td>
<td></td>
<td><strong>Aff0</strong></td>
</tr>
</tbody>
</table>

**Interrupt_Routing_Mode**, bit [31]

Interrupt_Routing_Mode. Defines how SPIs are routed in an affinity hierarchy:

<table>
<thead>
<tr>
<th>Interrupt_Routing_Mode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Interrupts routed to the PE specified by a.b.c.d. In this routing, a, b, c, and d are the values of fields Aff3, Aff2, Aff1, and Aff0 respectively.</td>
</tr>
<tr>
<td>0b1</td>
<td>Interrupts routed to any PE defined as a participating node.</td>
</tr>
</tbody>
</table>

If GICD_IROUTER<n>E.IRM == 0 and the affinity path does not correspond to an implemented PE, then if the corresponding interrupt becomes pending it will not be forwarded to any PE and will remain pending.

In implementations that do not require 1 of N distribution of SPIs, this bit might be RAZ/WI.

When this bit is set to 1, GICD_IROUTER<n>E.{Aff3, Aff2, Aff1, Aff0} are **UNKNOWN**.
An implementation might choose to make the Aff<n> fields RO when this field is 1.

Bits [30:24]
Reserved, RES0.

Aff2, bits [23:16]
Affinity level 2, an intermediate affinity level field.

Aff1, bits [15:8]
Affinity level 1, an intermediate affinity level field.

Aff0, bits [7:0]
Affinity level 0, the most significant affinity level field.

For an SPI with INTID m:

- The corresponding GICD_IROUTER<n>E register number, n, is given by n = m.
- The offset of the GICD_IROUTER<n>E register is 0x6000 + 8n.

Accessing the GICD_IROUTER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IROUTER<n>E, the register is RES0.

When GICD_CTLR.DS==0, a register that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IROUTER<n>E can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x8000 + 8n</td>
<td>GICD_IROUTER&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICD_ISACTIVER<n> characteristics are:

**Purpose**
Activates the corresponding interrupt. These registers are used when saving and restoring GIC state.

**Configuration**
Some or all RW fields of this register have defined reset values. These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common. The number of implemented GICD_ISACTIVER<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ISACTIVER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8. Accessing GICD_ISACTIVER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

**Attributes**
GICD_ISACTIVER<n> is a 32-bit register.

**Field descriptions**
The GICD_ISACTIVER<n> bit assignments are:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
| Set_active_bit<x>, bit [x], for x = 0 to 31 |

**Set_active_bit<x>, bit [x], for x = 0 to 31**
Adds the active state to interrupt number 32n + x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_active_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or is active and pending. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISACTIVER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ISACTIVER is (0x300 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.
Accessing the GICD_ISACTIVER<n>

When affinity routing is enabled for the Security state of an interrupt, bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ISACTIVER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

The bit reads as one if the status of the interrupt is active or active and pending. GICD_ISPENDR<n> and GICD_ICPENDR<n> provide the pending status of the interrupt.

GICD ISACTIVER<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0300 + 4n</td>
<td>GICD ISACTIVER&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
**GICD_ISACTIVER\(<n>E\), Interrupt Set-Active Registers (extended SPI range), n = 0 - 31**

The GICD_ISACTIVER\(<n>E\) characteristics are:

**Purpose**

Adds the active state to the corresponding SPI in the extended SPI range.

**Configuration**

Some or all RW fields of this register have defined reset values.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ISACTIVER\(<n>E\) registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

**Attributes**

GICD_ISACTIVER\(<n>E\) is a 32-bit register.

**Field descriptions**

The GICD_ISACTIVER\(<n>E\) bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Set_active_bit<\(x\)>, bit [x], for x = 0 to 31 |

**Set_active_bit<\(x\)>, bit [x], for x = 0 to 31**

For the extended SPIs, adds the active state to interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_active_bit&lt;(x)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or active and pending on this PE. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISACTIVER\(<n>E\) number, n, is given by \(n = (m-4096) \text{ DIV } 32\).
- The offset of the required GICD_ISACTIVER\(<n>E\) is \((0x1A00 + (4*n))\).
- The bit number of the required group modifier bit in this register is \((m-4096) \text{ MOD } 32\).

**Accessing the GICD_ISACTIVER\(<n>E\)**

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISACTIVER\(<n>E\), the corresponding bit is RES0.
When \texttt{GICD_CTLR.DS==0}, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

\textbf{\texttt{GICD_ISACTIVER}\textless{}\textgreater{}E can be accessed through the memory-mapped interfaces:}

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>$0x1A00 + 4n$</td>
<td>\texttt{GICD_ISACTIVER}\textless{}\textgreater{}E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{GICD_CTLR.DS == 0b0} accesses to this register are \textbf{RW}.
- When \texttt{IsAccessSecure()} accesses to this register are \textbf{RW}.
- When \texttt{!IsAccessSecure()} accesses to this register are \textbf{RW}.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The GICD_ISENABLER characteristics are:

**Purpose**

Enables forwarding of the corresponding interrupt to the CPU interfaces.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ISENABLER registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ISENABLER0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ISENABLER0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

**Attributes**

GICD_ISENABLER is a 32-bit register.

**Field descriptions**

The GICD_ISENABLER bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Set_enable_bit&lt;x&gt;, bit [x], for x = 0 to 31</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Set_enable_bit<x>, bit [x], for x = 0 to 31**

For SPIs and PPIs, controls the forwarding of interrupt number 32n + x to the CPU interfaces. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_enable_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

For SGIs, the behavior of this bit is IMPLEMENTATION DEFINED.

This field resets to an architecturally **UNKNOWN** value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISENABLER number, n, is given by n = m DIV 32.
- The offset of the required GICD_ISENABLER is (0x100 + (4*n)).
The bit number of the required group modifier bit in this register is \( m \mod 32 \).

At start-up, and after a reset, a PE can use this register to discover which peripheral INTIDs the GIC supports. If \( \text{GICD_CTLR}.DS == 0 \) in a system that supports EL3, the PE must do this for the Secure view of the available interrupts, and Non-secure software running on the PE must do this discovery after the Secure software has configured interrupts as Group 0/Secure Group 1 and Non-secure Group 1.

### Accessing the GICD_ISENABLER\(<n>\)

For SGIs and PPIs:

- When \( \text{ARE} = 1 \) for the Security state of an interrupt, the field for that interrupt is \( \text{RES}0 \) and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by \( \text{GICR_ISENABLER0} \).

Bits corresponding to unimplemented interrupts are RAZ/WI.

When \( \text{GICD_CTLR}.DS == 0 \), bits corresponding to Group 0 or Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether implemented SGIs are permanently enabled, or can be enabled and disabled by writes to \( \text{GICD_ISENABLER}<n> \) and \( \text{GICD_ICENABLER}<n> \) where \( n = 0 \).

For SPIs and PPIs, each bit controls the forwarding of the corresponding interrupt from the Distributor to the CPU interfaces.

**GICD_ISENABLER\(<n>\) can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x100 + 4n</td>
<td>GICD_ISENABLER(&lt;n&gt;)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \( \text{GICD_CTLR.DS} == 0b0 \) accesses to this register are \( \text{RW} \).
- When \( \text{IsAccessSecure()} \) accesses to this register are \( \text{RW} \).
- When \( \text{!IsAccessSecure()} \) accesses to this register are \( \text{RW} \).
GICD_ISENABLER\(<n\>)E, Interrupt Set-Enable Registers, \(n = 0 - 31\)

The GICD_ISENABLER\(<n\>)E characteristics are:

**Purpose**

Enables forwarding of the corresponding SPI in the extended SPI range to the CPU interfaces.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICD_ISENABLER\(<n\>)E are RES0.

When \(\text{GICD_TYPER.ESPI==0}\), these registers are RES0.

When \(\text{GICD_TYPER.ESPI==1}\), the number of implemented GICD_ISENABLER\(<n\>)E registers is \((\text{GICD_TYPER.ESPI_range+1})\). Registers are numbered from 0.

**Attributes**

GICD_ISENABLER\(<n\>)E is a 32-bit register.

**Field descriptions**

The GICD_ISENABLER\(<n\>)E bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Set_enable_bit\(<x\>)**, bit \([x]\), for \(x = 0\) to \(31\)

For the extended SPI range, controls the forwarding of interrupt number \(x\) to the CPU interface. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_enable_bit(&lt;x&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID \(m\), when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISENABLER\(<n\>)E number, \(n\), is given by \(n = (m-4096) \div 32\).
- The offset of the required GICD_ISENABLER\(<n\>)E is \((0\times1200 + (4\times n))\).
- The bit number of the required group modifier bit in this register is \((m-4096) \mod 32\).
**Accessing the GICD_ISENABLER\(<n>E**

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISENABLER\(<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICD_ISENABLER\(<n>E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x1200 + 4n</td>
<td>GICD_ISENABLER(&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICD_ISPENDR<n> characteristics are:

**Purpose**

Adds the pending state to the corresponding interrupt.

**Configuration**

Some or all RW fields of this register have defined reset values.

These registers are available in all GIC configurations. If GICD_CTLR.DS==0, these registers are Common.

The number of implemented GICD_ISPENDR<n> registers is (GICD_TYPER.ITLinesNumber+1). Registers are numbered from 0.

GICD_ISPENDR0 is Banked for each connected PE with GICR_TYPER.Processor_Number < 8.

Accessing GICD_ISPENDR0 from a PE with GICR_TYPER.Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

**Attributes**

GICD_ISPENDR<n> is a 32-bit register.

**Field descriptions**

The GICD_ISPENDR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Set_pending_bit<x>, bit [x], for x = 0 to 31 |

For SPIs and PPIs, adds the pending state to interrupt number 32n + x. Reads and writes have the following behavior:
**Set_pending_bit<x>**

<table>
<thead>
<tr>
<th>Set_pending_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending on any PE. If written, has no effect.</td>
</tr>
</tbody>
</table>
| 0b1                | If read, indicates that the corresponding interrupt is pending, or active and pending:  
|                    | • On this PE if the interrupt is an SGI or PPI.  
|                    | • On at least one PE if the interrupt is an SPI.  
|                    | If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases:  
|                    | • If the interrupt is an SGI. The pending state of an SGI can be set using **GICD_SPENDSGIR<n>**.  
|                    | • If the interrupt is not inactive and is not active.  
|                    | • If the interrupt is already pending because of a write to **GICD_ISPENDR<n>**.  
|                    | • If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.  
|                    | This field resets to 0. |

**Accessing the GICD_ISPENDR<n>**

Set-pending bits for SGIs are read-only and ignore writes. The Set-pending bits for SGIs are provided as **GICD_SPENDSGIR<n>**.

When affinity routing is enabled for the Security state of an interrupt:

- Bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by **GICR_ISPENDR0**.
- Bits corresponding to Group 0 and Group 1 Secure interrupts can only be set by Secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If **GICD_CTLR.DS==0**, unless the **GICD_NSACR<n>** registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

**GICD_ISPENDR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0200 + 4n</td>
<td>GICD_ISPENDR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_ISPENDR<n>E, Interrupt Set-Pending Registers (extended SPI range), n = 0 - 31

The GICD_ISPENDR<n>E characteristics are:

**Purpose**

Adds the pending state to the corresponding SPI in the extended SPI range.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICD_ISPENDR<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ISPENDR<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

**Attributes**

GICD_ISPENDR<n>E is a 32-bit register.

**Field descriptions**

The GICD_ISPENDR<n>E bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | Set_pending_bit<x>, bit [x], for x = 0 to 31 |

**Set_pending_bit<x>, bit [x], for x = 0 to 31**

For the extended SPIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_pending_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt frominactive to pending, or from active to active and pending. This has no effect in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is already pending because of a write to GICD_ISPENDR&lt;n&gt;E.</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:
• The corresponding GICD_ISPENDR\langle n\rangle E number, n, is given by \( n = (m - 4096) \div 32 \).
• The offset of the required GICD_ISPENDR\langle n\rangle E is \( (0x1600 + (4*n)) \).
• The bit number of the required group modifier bit in this register is \( (m - 4096) \mod 32 \).

**Accessing the GICD_ISPENDR\langle n\rangle E**

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISPENDR\langle n\rangle E, the corresponding bit is \texttt{RES0}.

When \texttt{GICD_CTLR.DS==0}, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICD_ISPENDR\langle n\rangle E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x1600 + 4n</td>
<td>GICD_ISPENDR\langle n\rangle E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When \texttt{GICD_CTLR.DS == 0b0} accesses to this register are \texttt{RW}.
• When \texttt{IsAccessSecure()} accesses to this register are \texttt{RW}.
• When \texttt{!IsAccessSecure()} accesses to this register are \texttt{RW}.
**Purpose**

When affinity routing is not enabled, holds the list of target PEs for the interrupt. That is, it holds the list of CPU interfaces to which the Distributor forwards the interrupt if it is asserted and has sufficient priority.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

These registers are available in all configurations of the GIC. When **GICD_CTLR.DS==0**, these registers are Common.

The number of implemented **GICD_ITARGETSR<n>** registers is 8*(**GICD_TYPER.ITLinesNumber**+1). Registers are numbered from 0.

**GICD_ITARGETSR0** to **GICD_ITARGETSR7** are Banked for each connected PE with **GICR_TYPER.Processor_Number < 8**.

Accessing **GICD_ITARGETSR0** to **GICD_ITARGETSR7** from a PE with **GICR_TYPER.Processor_Number > 7** is **CONSTRAINED UNPREDICTABLE**:

- Register is RAZ/WI.
- An **UNKNOWN** banked copy of the register is accessed.

**Attributes**

**GICD_ITARGETSR<n>** is a 32-bit register.

**Field descriptions**

The **GICD_ITARGETSR<n>** bit assignments are:

<table>
<thead>
<tr>
<th>Bit Offset</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>CPU_targets_offset_3B CPU_targets_offset_2B CPU_targets_offset_1B CPU_targets_offset_0B</td>
</tr>
</tbody>
</table>

PEs in the system number from 0, and each bit in a PE targets field refers to the corresponding PE. For example, a value of $0x3$ means that the Pending interrupt is sent to PEs 0 and 1. For **GICD_ITARGETSR0-GICD_ITARGETSR7**, a read of any targets field returns the number of the PE performing the read.

**CPU_targets_offset_3B**, bits [31:24]

PE targets for an interrupt, at byte offset 3.

This field resets to an architecturally **UNKNOWN** value.

**CPU_targets_offset_2B**, bits [23:16]

PE targets for an interrupt, at byte offset 2.

This field resets to an architecturally **UNKNOWN** value.

**CPU_targets_offset_1B**, bits [15:8]

PE targets for an interrupt, at byte offset 1.
This field resets to an architecturally **UNKNOWN** value.

**CPU_targets_offset_0B, bits [7:0]**

PE targets for an interrupt, at byte offset 0.

This field resets to an architecturally **UNKNOWN** value.

The bits that are set to 1 in the PE targets field determine which PEs are targeted:

<table>
<thead>
<tr>
<th>Value of PE targets field</th>
<th>Interrupt targets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0bxxxxxxx1</td>
<td>CPU interface 0</td>
</tr>
<tr>
<td>0bxxxxxx1x</td>
<td>CPU interface 1</td>
</tr>
<tr>
<td>0bxxxxx1xx</td>
<td>CPU interface 2</td>
</tr>
<tr>
<td>0bxxxx1xxx</td>
<td>CPU interface 3</td>
</tr>
<tr>
<td>0bxxx1xxxx</td>
<td>CPU interface 4</td>
</tr>
<tr>
<td>0bxx1xxxxx</td>
<td>CPU interface 5</td>
</tr>
<tr>
<td>0bx1xxxxxx</td>
<td>CPU interface 6</td>
</tr>
<tr>
<td>0b1xxxxxxx</td>
<td>CPU interface 7</td>
</tr>
</tbody>
</table>

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD ITARGETSR<n> number, n, is given by n = m DIV 4.
- The offset of the required GICD_ITARGETSR<n> register is (0x800 + (4*n)).
- The byte offset of the required Priority field in this register is m MOD 4, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

Software can write to these registers at any time. Any change to a targets field value:

- Has no effect on any active interrupt. This means that removing a CPU interface from a targets list does not cancel an active state for interrupts on that CPU interface. There is no effect on interrupts that are active and pending until the active status is cleared, at which time it is treated as a pending interrupt.
- Has an effect on any pending interrupts. This means:
  - Enables the CPU interface to be chosen as a target for the pending interrupt using an **IMPLEMENTATION DEFINED** mechanism.
  - Removing a CPU interface from the target list of a pending interrupt removes the pending state of the interrupt on that CPU interface.

**Accessing the GICD_ITARGETSR<n>**

These registers are used when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt, the target PEs for an interrupt are defined by GICD_IROUTER<n> and the associated byte in GICD_ITARGETSR<n> is RES0. An implementation is permitted to make the byte RAZ/WI in this case.

- These registers are byte-accessible.
- A register field corresponding to an unimplemented interrupt is RAZ/WI.
- A field bit corresponding to an unimplemented CPU interface is RAZ/WI.
- GICD_ITARGETSR0-GICD_ITARGETSR7 are read-only. Each field returns a value that corresponds only to the PE reading the register.
- It is **IMPLEMENTATION DEFINED** which, if any, SPIs are statically configured in hardware. The field for such an SPI is read-only, and returns a value that indicates the PE targets for the interrupt.
- If GICD_CTLR.DS==0, unless the GICD_NSACR<n> registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

In a single connected PE implementation, all interrupts target one PE, and these registers are RAZ/WI.

**Note**

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.
GICD_ITARGETSR<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0000 + 4n</td>
<td>GICD_ITARGETSR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GICD_NSACR<n>, Non-secure Access Control Registers, n = 0 - 63

The GICD_NSACR<n> characteristics are:

**Purpose**

Enables Secure software to permit Non-secure software on a particular PE to create and control Group 0 interrupts.

**Configuration**

Some or all RW fields of this register have defined reset values.

The concept of selective enabling of Non-secure access to Group 0 and Secure Group 1 interrupts applies to SGIs and SPIs.

GICD_NSACR0 is a Banked register used for SGIs. A copy is provided for every PE that has a CPU interface and that supports this feature.

**Attributes**

GICD_NSACR<n> is a 32-bit register.

**Field descriptions**

The GICD_NSACR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td><strong>NS_access&lt;x&gt;, bits [2x+1:2x], for x = 0 to 15</strong></td>
</tr>
</tbody>
</table>

**NS_access<x>, bits [2x+1:2x], for x = 0 to 15**

Controls Non-secure access of the interrupt with ID 16n + x.

If the corresponding interrupt does not support configurable Non-secure access, the field is RAZ/WI.

Otherwise, the field is RW and determines the level of Non-secure control permitted if the interrupt is a Secure interrupt. If the interrupt is a Non-secure interrupt, this field is ignored.

The possible values of each 2-bit field are:
**GICD_NSACR<n>, Non-secure Access Control Registers, n = 0 - 63**

<table>
<thead>
<tr>
<th>NS_access&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>No Non-secure access is permitted to fields associated with the corresponding interrupt.</td>
</tr>
<tr>
<td>0b01</td>
<td>Non-secure read and write access is permitted to set-pending bits in <strong>GICD_ISPENDR&lt;n&gt;</strong> associated with the corresponding interrupt. A Non-secure write access to <strong>GICD_SETSPI_NSR</strong> is permitted to set the pending state of the corresponding interrupt. A Non-secure write access to <strong>GICD_SGIR</strong> is permitted to generate a Secure SGI for the corresponding interrupt. An implementation might also provide read access to clear-pending bits in <strong>GICD_ICPENDR&lt;n&gt;</strong> associated with the corresponding interrupt.</td>
</tr>
<tr>
<td>0b10</td>
<td>As 0b01, but adds Non-secure read and write access permission to fields associated with the corresponding interrupt in the <strong>GICD_ICPENDR&lt;n&gt;</strong> registers. A Non-secure write access to <strong>GICD_CLRSPINSR</strong> is permitted to clear the pending state of the corresponding interrupt. Also adds Non-secure read access permission to fields associated with the corresponding interrupt in the <strong>GICD_ISACTIVER&lt;n&gt;</strong> and <strong>GICD_ICACTIVER&lt;n&gt;</strong> registers.</td>
</tr>
<tr>
<td>0b11</td>
<td>For GICD_NSACR0 this encoding is reserved and treated as 10. For all other GICD_NSACR&lt;n&gt; registers this encoding is treated as 0b10, but adds Non-secure read and write access permission to <strong>GICD_ITARGETSR&lt;n&gt;</strong> and <strong>GICD_IROUTER&lt;n&gt;</strong> fields associated with the corresponding interrupt.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_NSACR<n> number, n, is given by n = m DIV 16.
- The offset of the required GICD_NSACR<n> register is (0xE00 + (4*n)).

**Note**

Because each field in this register comprises two bits, GICD_NSACR0 controls access rights to SGI registers, GICD_NSACR1 controls access to PPI registers (and is always RAZ/WI), and all other GICD_NSACR<n> registers control access to SPI registers.

For compatibility with GICv2, writes to GICD_NSACR0 for a particular PE must be coordinated within the Distributor and must update **GICR_NSACR** for the Redistributor associated with that PE.

**Accessing the GICD_NSACR<n>**

When **GICD_CTLR.DS==1**, this register is RAZ/WI.

These registers are Secure, and are RAZ/WI to Non-secure accesses.

These registers are always used when affinity routing is not enabled. When affinity routing is enabled for the Secure state, GICD_NSACR0 is RES0 and **GICR_NSACR** provides equivalent functionality for SGIs.

These registers do not support PPIs, therefore GICD_NSACR1 is RAZ/WI.

**GICD_NSACR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0E00 + 4n</td>
<td>GICD_NSACR&lt;n&gt;</td>
</tr>
</tbody>
</table>

When When IsAccessSecure() access on this interface are **RW**.
GICD_NSACR\(n\)E, Non-secure Access Control Registers, \(n = 0 - 63\)

The GICD_NSACR\(n\)E characteristics are:

**Purpose**

Enables Secure software to permit Non-secure software on a particular PE to create and control Group 0 interrupts.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, \(>=3.1\) is implemented. Otherwise, direct accesses to GICD_NSACR\(n\)E are

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
</tr>
</tbody>
</table>

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICFGR\(n\)E registers is \(((GICD_TYPER.ESPI\_range+1)*2)\). Registers are numbered from 0.

**Attributes**

GICD_NSACR\(n\)E is a 32-bit register.

**Field descriptions**

The GICD_NSACR\(n\)E bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NS_access(x), bits [2x+1:2x], for (x = 0) to (15)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
</tbody>
</table>

**NS_access\(x\), bits [2x+1:2x], for \(x = 0\) to \(15\)**

Controls Non-secure access of the interrupt with ID 16\(n\) + \(x\).

If the corresponding interrupt does not support configurable Non-secure access, the field is RAZ/WI.

Otherwise, the field is RW and determines the level of Non-secure control permitted if the interrupt is a Secure interrupt. If the interrupt is a Non-secure interrupt, this field is ignored.

The possible values of each 2-bit field are:
Meaning
0b00 No Non-secure access is permitted to fields associated with the corresponding interrupt.
0b01 Non-secure read and write access is permitted to set-pending bits in GICD_ISPENDR<n>E associated with the corresponding interrupt. A Non-secure write access to GICD_SETSPI_NS is permitted to set the pending state of the corresponding interrupt.
0b10 As 0b01, but adds Non-secure read and write access permission to fields associated with the corresponding interrupt in the GICD_ICPENDR<n>E registers. A Non-secure write access to GICD_CLR_SPI_NS is permitted to clear the pending state of the corresponding interrupt. Also adds Non-secure read access permission to fields associated with the corresponding interrupt in the GICD_ISACTIVER<n>E and GICD_ICACTIVER<n>E registers.
0b11 This encoding is treated as 0b10, but adds Non-secure read and write access permission to GICD_IROUTER<n>E fields associated with the corresponding interrupt.

This field resets to 0.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_NSACR<n>E number, n, is given by n = (m - 4096) DIV 16.
- The offset of the required GICD_NSACR<n>E register is (0x3600 + (4*n)).

### Accessing the GICD_NSACR<n>E

**GICD_NSACR<n>E** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x3600 + 4n</td>
<td>GICD_NSACR&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RAZ/WI.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RAZ/WI.
**GICD_SETSPI_NSR, Set Non-secure SPI Pending Register**

The GICD_SETSPI_NSR characteristics are:

**Purpose**

Adds the pending state to a valid SPI if permitted by the Security state of the access and the GICD_NSACR<n> value for that SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

**Configuration**

If GICD_TYPER.MBIS == 0, this register is reserved.

When GICD_CTLR.DS==1, this register provides functionality for all SPIs.

**Attributes**

GICD_SETSPI_NSR is a 32-bit register.

**Field descriptions**

The GICD_SETSPI_NSR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    | RES0|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|    |    |    |    |    |    |    | INTID|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:13]**

Reserved, RES0.

**INTID, bits [12:0]**

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to GICD_CLRSP1_NSR, GICD_CLRSP1_SR, or GICD_ICPENDR<n>.
- For a level-sensitive interrupt, a write to GICD_SETSPI_NSR or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to GICD_CLRSP1_NSR or GICD_CLRSP1_SR. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

**Accessing the GICD_SETSPI_NSR**

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding GICD_NSACR<n> register is 0.
- The value written specifies an invalid SPI.
- The SPI is already pending.
16-bit accesses to bits [15:0] of this register must be supported.

---

**Note**

A Secure access to this register can set the pending state of any valid SPI.

---

**GICD_SETSPI_NSR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0040</td>
<td>GICD_SETSPI_NSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
The GICD_SETSPI_SR characteristics are:

**Purpose**

Adds the pending state to a valid SPI. A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

**Configuration**

If `GICD_TYPER_MBIS` == 0, this register is reserved. When `GICD_CTLR_DS` == 1, this register is WI.

**Attributes**

GICD_SETSPI_SR is a 32-bit register.

**Field descriptions**

The GICD_SETSPI_SR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:13]**

Reserved, RES0.

**INTID, bits [12:0]**

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to `GICD_SETSPI_NSR` or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to `GICD_CLRSPI_NSR`, `GICD_CLRSPI_SR`, or `GICD_ICPENDR<n>`.
- For a level-sensitive interrupt, a write to `GICD_SETSPI_NSR` or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to `GICD_CLRSPI_NSR` or `GICD_CLRSPI_SR`. If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

**Accessing the GICD_SETSPI_SR**

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.
GICD_SETSPI_SR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0050</td>
<td>GICD_SETSPI_SR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WI**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WI**.
The GICD_SGIR characteristics are:

**Purpose**

Controls the generation of SGIs.

**Configuration**

This register is available in all configurations of the GIC. If the GIC supports two Security states this register is Common.

**Attributes**

GICD_SGIR is a 32-bit register.

**Field descriptions**

The GICD_SGIR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | TargetListFilter | CPUTargetList | NSATT | RES0 | INTID |

**Bits [31:26]**

Reserved, RES0.

**TargetListFilter, bits [25:24]**

Determines how the Distributor processes the requested SGI.

<table>
<thead>
<tr>
<th>TargetListFilter</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Forward the interrupt to the CPU interfaces specified by GICD_SGIR.CPUTargetList.</td>
</tr>
<tr>
<td>0b01</td>
<td>Forward the interrupt to all CPU interfaces except that of the PE that requested the interrupt.</td>
</tr>
<tr>
<td>0b10</td>
<td>Forward the interrupt only to the CPU interface of the PE that requested the interrupt.</td>
</tr>
<tr>
<td>0b11</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**CPUTargetList, bits [23:16]**

When GICD_SGIR.TargetListFilter is 0b00, this field defines the CPU interfaces to which the Distributor must forward the interrupt.

Each bit of the field refers to the corresponding CPU interface. For example, CPUTargetList[0] corresponds to interface 0. Setting a bit to 1 indicates that the interrupt must be forwarded to the corresponding interface.

If this field is 0b00000000 when GICD_SGIR.TargetListFilter is 0b00, the Distributor does not forward the interrupt to any CPU interface.

**NSATT, bit [15]**

Specifies the required group of the SGI.
<table>
<thead>
<tr>
<th>NSATT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forward the SGI specified in the INTID field to a specified CPU interface only if the SGI is configured as Group 0 on that interface.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forward the SGI specified in the INTID field to a specified CPU interface only if the SGI is configured as Group 1 on that interface.</td>
</tr>
</tbody>
</table>

This field is writable only by a Secure access. Non-secure accesses can also generate Group 0 interrupts, if allowed to do so by GICD_NSACR0. Otherwise, Non-secure writes to GICD_SGIR generate an SGI only if the specified SGI is programmed as Group 1, regardless of the value of bit [15] of the write.

### Bits [14:4]

Reserved, RES0.

### INTID, bits [3:0]

The INTID of the SGI to forward to the specified CPU interfaces.

### Accessing the GICD_SGIR

This register is used only when affinity routing is not enabled. When affinity routing is enabled, this register is RES0.

It is **IMPLEMENTATION DEFINED** whether this register has any effect when the forwarding of interrupts by the Distributor is disabled by GICD_CTLR.

**GICD_SGIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0F00</td>
<td>GICD_SGIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
The GICD_SPENDSGIR<n> characteristics are:

**Purpose**

Adds the pending state to an SGI.

A write to this register changes the state of an inactive SGI to pending, and the state of an active SGI to active and pending.

**Configuration**

Some or all RW fields of this register have defined reset values.

Four SGI set-pending registers are implemented. Each register contains eight set-pending bits for each of four SGIs, for a total of 16 possible SGIs.

In multiprocessor implementations, each PE has a copy of these registers.

**Attributes**

GICD_SPENDSGIR<n> is a 32-bit register.

**Field descriptions**

The GICD_SPENDSGIR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>(0x0) If read, indicates that the SGI from the corresponding PE is not pending and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0</td>
<td>(0x0) If read, indicates that the SGI from the corresponding PE is pending or is active and pending. If written, adds the pending state to the SGI for the corresponding PE.</td>
</tr>
</tbody>
</table>

This field resets to \(0\).

For SGI ID \(m\), generated by processing element \(C\) writing to the corresponding GICD_SGIR field, where DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_SPENDSGIR<n> number is given by \(n = m \text{ DIV } 4\).
- The offset of the required register is \((0xF20 + (4n))\).
- The offset of the required field within the register GICD_SPENDSGIR<n> is given by \(m \text{ MOD } 4\).
- The required bit in the 8-bit SGI set-pending field \(m\) is bit \(C\).
**Accessing the GICD_SPENDSGIR<n>**

These registers are used only when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt then the bit associated with SGI in that Security state is RES0. An implementation is permitted to make the register RAZ/WI in this case.

A register bit that corresponds to an unimplemented SGI is RAZ/WI.

These registers are byte-accessible.

If the GIC implementation supports two Security states:

- A register bit that corresponds to a Group 0 interrupt is RAZ/WI to Non-secure accesses.
- Register bits corresponding to unimplemented PEs are RAZ/WI.

**GICD_SPENDSGIR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0F20 + 4n</td>
<td>GICD_SPENDSGIR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICD_STATUSR characteristics are:

**Purpose**

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

**Configuration**

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

**Attributes**

GICD_STATUSR is a 32-bit register.

**Field descriptions**

The GICD_STATUSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>WROD</td>
<td>Write to an RO location.</td>
</tr>
<tr>
<td>29</td>
<td>RWOD</td>
<td>Read of a WO location.</td>
</tr>
<tr>
<td>28</td>
<td>WRD</td>
<td>Write to a reserved location.</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, RES0.

**WROD, bit [3]**

Write to an RO location.

<table>
<thead>
<tr>
<th>WROD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write to an RO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**RWOD, bit [2]**

Read of a WO location.

<table>
<thead>
<tr>
<th>RWOD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a WO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**WRD, bit [1]**

Write to a reserved location.
<table>
<thead>
<tr>
<th>WRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write to a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**RRD, bit [0]**

Read of a reserved location.

<table>
<thead>
<tr>
<th>RRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**Accessing the GICD_STATUSR**

This is an optional register. If the register is not implemented, the location is RAZ/WI.

**GICD_STATUSR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0010</td>
<td>GICD_STATUSR(S)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0010</td>
<td>GICD_STATUSR(NS)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICD_TYPER, Interrupt Controller Type Register

The GICD_TYPER characteristics are:

### Purpose

Provides information about what features the GIC implementation supports. It indicates:

- Whether the GIC implementation supports two Security states.
- The maximum number of INTIDs that the GIC implementation supports.
- The number of PEs that can be used as interrupt targets.

### Configuration

This register is available in all configurations of the GIC. When `GICD_CTLR.DS==0`, this register is Common.

### Attributes

GICD_TYPER is a 32-bit register.

### Field descriptions

The GICD_TYPER bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Bit Assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESPI_range</td>
<td>[31:27]</td>
</tr>
<tr>
<td>RSS</td>
<td>[26]</td>
</tr>
<tr>
<td>No1N</td>
<td>[25]</td>
</tr>
<tr>
<td>IDbits</td>
<td>[24]</td>
</tr>
<tr>
<td>DVIS</td>
<td>[23]</td>
</tr>
<tr>
<td>LPIS</td>
<td>[22]</td>
</tr>
<tr>
<td>MBIS</td>
<td>[21]</td>
</tr>
<tr>
<td>num_LPIs</td>
<td>[20]</td>
</tr>
<tr>
<td>SecurityExtn</td>
<td>[19]</td>
</tr>
<tr>
<td>RES0</td>
<td>[18]</td>
</tr>
<tr>
<td>ESP1</td>
<td>[17]</td>
</tr>
<tr>
<td>CPUNumber</td>
<td>[16]</td>
</tr>
<tr>
<td>ITLinesNumber</td>
<td>[15:0]</td>
</tr>
</tbody>
</table>

#### ESPI_range field, bits [31:27]

When GICD_TYPER.ESPI == 1:

Indicates the maximum INTID in the Extended SPI range.

- Maximum Extended SPI INTID is \((32 \times (\text{ESPI\_range} + 1) + 4095)\)

Otherwise:

Reserved, RES0.

#### RSS, bit [26]

Range Selector Support. Possible values are:

<table>
<thead>
<tr>
<th>RSS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The IRI supports targeted SGIs with affinity level 0 values of 0 - 15.</td>
</tr>
<tr>
<td>0b1</td>
<td>The IRI supports targeted SGIs with affinity level 0 values of 0 - 255.</td>
</tr>
</tbody>
</table>

#### No1N, bit [25]

Indicates whether 1 of N SPI interrupts are supported.

<table>
<thead>
<tr>
<th>No1N</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>1 of N SPI interrupts are supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>1 of N SPI interrupts are not supported.</td>
</tr>
</tbody>
</table>
A3V, bit [24]

Affinity 3 valid. Indicates whether the Distributor supports nonzero values of Affinity level 3. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Distributor only supports zero values of Affinity level 3.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Distributor supports nonzero values of Affinity level 3.</td>
</tr>
</tbody>
</table>

IDbits, bits [23:19]

The number of interrupt identifier bits supported, minus one.

DVIS, bit [18]

Indicates whether the implementation supports Direct Virtual LPI injection.

<table>
<thead>
<tr>
<th>DVIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support Direct Virtual LPI injection.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports Direct Virtual LPI injection.</td>
</tr>
</tbody>
</table>

For GICv3, this field is RES0.

LPIS, bit [17]

Indicates whether the implementation supports LPIs.

<table>
<thead>
<tr>
<th>LPIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support LPIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports LPIs.</td>
</tr>
</tbody>
</table>

MBIS, bit [16]

Indicates whether the implementation supports message-based interrupts by writing to Distributor registers.

<table>
<thead>
<tr>
<th>MBIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support message-based interrupts by writing to Distributor registers. The GICD_CLR_SPI_NSR, GICD_SET_SPI_NSR, GICD_CLR_SPI_SR, and GICD_SET_SPI_SR registers are reserved.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports message-based interrupts by writing to the GICD_CLR_SPI_NSR, GICD_SET_SPI_NSR, GICD_CLR_SPI_SR, or GICD_SET_SPI_SR registers.</td>
</tr>
</tbody>
</table>

num_LPIs, bits [15:11]

Number of supported LPIs.

- 0b00000 Number of LPIs as indicated by GICD_TYPER.IDbits.
- All other values Number of LPIs supported is \( 2^{\text{num}_\text{LPIs}+1} \).
  - Available LPI INTIDs are 8192..(8192 + 2^{\text{num}_\text{LPIs}+1} - 1).
  - This field cannot indicate a maximum LPI INTID greater than that indicated by GICD_TYPER.IDbits.

When the supported INTID width is less than 14 bits, this field is RES0 and no LPIs are supported.

SecurityExtn, bit [10]

Indicates whether the GIC implementation supports two Security states:

When GICD_CTRLR.DS == 1, this field is RAZ.
<table>
<thead>
<tr>
<th>SecurityExtn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The GIC implementation supports only a single Security state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The GIC implementation supports two Security states.</td>
</tr>
</tbody>
</table>

**Bit [9]**

Reserved, RES0.

**ESPI, bit [8]**

Extended SPI

<table>
<thead>
<tr>
<th>ESPI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Extended SPI range not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Extended SPI range implemented.</td>
</tr>
</tbody>
</table>

**CPUNumber, bits [7:5]**

Reports the number of PEs that can be used when affinity routing is not enabled, minus 1.

These PEs must be numbered contiguously from zero, but the relationship between this number and the affinity hierarchy from MPIDR is IMPLEMENTATION DEFINED. If the implementation does not support ARE being zero, this field is 000.

**ITLinesNumber, bits [4:0]**

Indicates the maximum SPI INTID that the GIC implementation supports. If the value of this field is N, the maximum SPI INTID is 32(N+1)-1. For example, 00011 specifies that the maximum SPI INTID is 127.

The maximum SPI INTID an implementation might support is 1019 (field value 11111). Regardless of the range of INTIDs defined by this field, interrupt IDs 1020-1023 are reserved for special purposes.

A value of 0 indicates no SPIs are support.

**Note**

The value derived from this field specifies the maximum number of SPIs that the GIC implementation might support. An implementation might not implement all SPIs up to this maximum.

The ITLinesNumber field only indicates the maximum number of SPIs that the GIC implementation might support. This value determines the number of instances of the following interrupt registers:

- GICD_IGROUPR<n>
- GICD_ISENABLER<n>
- GICD_ICENABLER<n>
- GICD_ISPENDR<n>
- GICD_ICPENDR<n>
- GICD_ISACTIVER<n>
- GICD_ICACTIVER<n>
- GICD_IPRIORITYR<n>
- GICD_ITARGETSR<n>
- GICD_ICFGR<n>

The GIC architecture does not require a GIC implementation to support a continuous range of SPI interrupt IDs. Software must check which SPI INTIDs are supported, up to the maximum value indicated by GICD_TYPER.ITLinesNumber.
Accessing the GICD_TYPER

GICD_TYPER can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Distributor</td>
<td>0x0004</td>
<td>GICD_TYPER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The GICH_APR<n> characteristics are:

**Purpose**

These registers track which preemption levels are active in the virtual CPU interface, and indicate the current active priority. Corresponding bits are set to 1 in this register when an interrupt is acknowledged, based on GICH_LR<n>.Priority, and the least significant bit set is cleared on EOI.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is available when the GIC implementation supports interrupt virtualization.

The number of registers required depends on how many bits are implemented in GICH_LR<n>.Priority:

- When 5 priority bits are implemented, 1 register is required (GICH_APR0).
- When 6 priority bits are implemented, 2 registers are required (GICH_APR0, GICH_APR1).
- When 7 priority bits are implemented, 4 registers are required (GICH_APR0, GICH_APR1, GICH_APR2, GICH_APR3).

Unimplemented registers are RAZ/WI.

**Attributes**

GICH_APR<n> is a 32-bit register.

**Field descriptions**

The GICH_APR<n> bit assignments are:

<table>
<thead>
<tr>
<th>P&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no interrupt active at the priority corresponding to that bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>There is an interrupt active at the priority corresponding to that bit.</td>
</tr>
</tbody>
</table>

Active priorities. Possible values of each bit are:

If 5 bits of priority are implemented (bits [7:3] of priority), then there are 32 priority groups, and the active state of these priorities are held in GICH_APR0 in the bits corresponding to Priority[7:3].

If 6 bits of priority are implemented (bits [7:2] of priority), then there are 64 priority groups, and:

- The active state of priorities 0 - 124 are held in GICH_APR0 in the bits corresponding to 0:Priority[6:2].
- The active state of priorities 128 - 252 are held in GICH_APR1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of priority are implemented (bits [7:1] of priority), then there are 128 priority groups, and:

- The active state of priorities 0 - 62 are held in GICH_APR0 in the bits corresponding to 0:Priority[5:1].
- The active state of priorities 64 - 126 are held in GICH_APR1 in the bits corresponding to 01:Priority[5:1].
- The active state of priorities 128 - 190 are held in GICH_APR2 in the bits corresponding to 10:Priority[5:1].
The active state of priorities 192 - 254 are held in GICH_APR3 in the bits corresponding to 11:Priority[5:1]. This field resets to 0.

**Accessing the GICH_APR<n>**

These registers are used only when System register access is not enabled. When System register access is enabled the following registers provide equivalent functionality:

- **In AArch64:**
  - For Group 0, `ICH_AP0R<n>_EL2`.
  - For Group 1, `ICH_AP1R<n>_EL2`.
- **In AArch32:**
  - For Group 0, `ICH_AP0R<n>`.
  - For Group 1, `ICH_AP1R<n>`.

**GICH_APR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x00F0 + 4n</td>
<td>GICH_APR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICH_EISR, End Interrupt Status Register

The GICH_EISR characteristics are:

**Purpose**

Indicates which List registers have outstanding EOI maintenance interrupts.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICH_EISR is a 32-bit register.

**Field descriptions**

The GICH_EISR bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | Status<n>, bit [n], for n = 0 to 15 |

**Bits [31:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 0 to 15**

EOI maintenance interrupt status for List register <n>:

<table>
<thead>
<tr>
<th>Status&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICH_LR&lt;n&gt; does not have an EOI maintenance interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICH_LR&lt;n&gt; has an EOI maintenance interrupt that has not been handled.</td>
</tr>
</tbody>
</table>

For any GICH_LR<n> register, the corresponding status bit is set to 1 if all of the following are true:

- GICH_LR<n>.State is 0b00.
- GICH_LR<n>.HW == 0.
- GICH_LR<n>.EOI == 1.

This field resets to an architecturally UNKNOWN value.

**Accessing the GICH_EISR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_EISR provides equivalent functionality.
- For AArch64 implementations, ICH_EISR_EL2 provides equivalent functionality.

Bits corresponding to unimplemented List registers are RAZ.
GICH_EISR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0020</td>
<td>GICH_EISR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The GICH_ELRSR characteristics are:

**Purpose**

Indicates which List registers contain valid interrupts.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICH_ELRSR is a 32-bit register.

**Field descriptions**

The GICH_ELRSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>RES0</td>
</tr>
<tr>
<td>31-15</td>
<td>Status&lt;n&gt;, bit [n], for n = 0 to 15</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**Status<n>, bit [n], for n = 0 to 15**

Status bit for List register <n>:

<table>
<thead>
<tr>
<th>Status&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td><strong>GICH_LR&lt;n&gt;</strong>, if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td><strong>GICH_LR&lt;n&gt;</strong> does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt.</td>
</tr>
</tbody>
</table>

For any **GICH_LR<n>** register, the corresponding status bit is set to 1 if **GICH_LR<n>**.State is 0b00 and either:

- **GICH_LR<n>**.HW == 1.
- **GICH_LR<n>**.EOI == 0.

This field resets to 1.

**Accessing the GICH_ELRSR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, **ICH_ELRSR** provides equivalent functionality.
- For AArch64 implementations, **ICH_ELRSR_EL2** provides equivalent functionality.

Bits corresponding to unimplemented List registers are RES0.
**GICH_ELRSR** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0030</td>
<td>GICH_ELRSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.
The GICH_HCR characteristics are:

**Purpose**

Controls the virtual CPU interface.

**Configuration**

RW fields in this register reset to architecturally \texttt{UNKNOWN} values.

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICH_HCR is a 32-bit register.

**Field descriptions**

The GICH_HCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-27</td>
<td>EOICount</td>
</tr>
<tr>
<td>26-8</td>
<td>Reserved, \texttt{RES0}</td>
</tr>
<tr>
<td>7</td>
<td>VGrp1DIE</td>
</tr>
<tr>
<td>6</td>
<td>VGrp1EIE</td>
</tr>
<tr>
<td>5</td>
<td>VGrp0DIE</td>
</tr>
<tr>
<td>4</td>
<td>VGrp0EIE</td>
</tr>
<tr>
<td>3</td>
<td>NPIE</td>
</tr>
<tr>
<td>2</td>
<td>LRENPIE</td>
</tr>
<tr>
<td>1</td>
<td>UIE</td>
</tr>
<tr>
<td>0</td>
<td>En</td>
</tr>
</tbody>
</table>

**EOICount, bits [31:27]**

Counts the number of EOs received that do not have a corresponding entry in the List registers. The virtual CPU interface increments this field automatically when a matching EOI is received. EOs that do not clear a bit in \texttt{GICH_APR<n>} do not cause an increment. If an EOI occurs when the value of this field is 31, then the field wraps to 0.

The maintenance interrupt is asserted whenever this field is nonzero and \texttt{GICH_HCR.LRENPIE == 1}.

This field resets to an architecturally \texttt{UNKNOWN} value.

**Bits [26:8]**

Reserved, \texttt{RES0}.

**VGrp1DIE, bit [7]**

VM Group 1 Disabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected virtual machine is disabled:

<table>
<thead>
<tr>
<th>VGrp1DIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when \texttt{GICV_CTLR.EnableGrp1 == 0}.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally \texttt{UNKNOWN} value.

**VGrp1EIE, bit [6]**

VM Group 1 Enabled Interrupt Enable.
Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected virtual machine is enabled:

<table>
<thead>
<tr>
<th>VGrp1EIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when</td>
</tr>
<tr>
<td></td>
<td>GICV_CTLR.EnableGrp1 == 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**VGrp0DIE, bit [5]**

VM Group 0 Disabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected virtual machine is disabled:

<table>
<thead>
<tr>
<th>VGrp0DIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when</td>
</tr>
<tr>
<td></td>
<td>GICV_CTLR.EnableGrp0 == 0.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**VGrp0EIE, bit [4]**

VM Group 0 Enabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected virtual machine is enabled:

<table>
<thead>
<tr>
<th>VGrp0EIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled when</td>
</tr>
<tr>
<td></td>
<td>GICV_CTLR.EnableGrp0 == 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**NPIE, bit [3]**

No Pending Interrupt Enable.

Enables the signaling of a maintenance interrupt while no pending interrupts are present in the List registers:

<table>
<thead>
<tr>
<th>NPIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled while the List registers contain no interrupts in the pending state.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**LRENPIE, bit [2]**

List Register Entry Not Present Interrupt Enable.

Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register for an EOI request:

<table>
<thead>
<tr>
<th>LRENPIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Maintenance interrupt signaled while GICV_HCR.EOICount is not 0.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.
UIE, bit [1]

Underflow Interrupt Enable.

Enables the signaling of a maintenance interrupt when the List registers are either empty or hold only one valid entry.

<table>
<thead>
<tr>
<th>UIE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Maintenance interrupt disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>A maintenance interrupt is signaled if zero or one of the List register entries are marked as a valid interrupt.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally `UNKNOWN` value.

En, bit [0]

Enable.

Global enable bit for the virtual CPU interface.

<table>
<thead>
<tr>
<th>En</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual CPU interface operation is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual CPU interface operation is enabled.</td>
</tr>
</tbody>
</table>

When this field is 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of `GICV_IAR` or `GICV_AIAR` returns a spurious interrupt ID.

This field resets to an architecturally `UNKNOWN` value.

The VGrp1DIE, VGrp1EIE, VGrp0DIE, and VGrp0EIE fields permit the hypervisor to track the virtual CPU interfaces that are enabled. The hypervisor can then route interrupts that have multiple targets correctly and efficiently, without having to read the virtual CPU interface status.

See Maintenance interrupts and `GICH_MISR` for more information.

**Accessing the GICH_HCR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, `ICH_HCR` provides equivalent functionality.
- For AArch64 implementations, `ICH_HCR_EL2` provides equivalent functionality.

GICH_HCR.En must be set to 1 for any virtual or maintenance interrupt to be asserted.

**GICH_HCR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0000</td>
<td>GICH_HCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICH_LR<n> characteristics are:

**Purpose**

These registers provide context information for the virtual CPU interface.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is available when the GIC implementation supports interrupt virtualization.

A maximum of 16 List registers can be provided. **GICH_VTR.ListRegs** defines the number implemented. Unimplemented List registers are RAZ/WI.

**Attributes**

GICH_LR<n> is a 32-bit register.

**Field descriptions**

The GICH_LR<n> bit assignments are:

<table>
<thead>
<tr>
<th>HW</th>
<th>Group</th>
<th>State</th>
<th>Priority</th>
<th>RES0</th>
<th>pINTID</th>
<th>vINTID</th>
</tr>
</thead>
</table>

**HW, bit [31]**

Indicates whether this virtual interrupt is a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt corresponding to the INTID:

<table>
<thead>
<tr>
<th>HW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This interrupt is triggered entirely in software. No notification is sent to the Distributor when the virtual interrupt is deactivated.</td>
</tr>
<tr>
<td>0b1</td>
<td>A hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using GICH_LR&lt;n&gt;.pINTID to indicate the physical interrupt identifier. If <strong>GICV_CTLR.EOlmode == 0</strong>, this request corresponds to a write to <strong>GICV_EOIR</strong> or <strong>GICV_AEOIR</strong>, otherwise it corresponds to a write to <strong>GICV_DIR</strong>.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Group, bit [30]**

Indicates whether the interrupt is Group 0 or Group 1:

<table>
<thead>
<tr>
<th>Group</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 virtual interrupt. <strong>GICV_CTLR.FIQEn</strong> determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and <strong>GICV_CTLR.EnableGrp0</strong> enables signaling of this interrupt to the virtual machine.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 virtual interrupt, signaled as a virtual IRQ. <strong>GICV_CTLR.EnableGrp1</strong> enables signaling of this interrupt to the virtual machine.</td>
</tr>
</tbody>
</table>

**Note**
CBPR controls whether GICV_BPR or GICV_ABPR determines if a pending Group 1 interrupt has sufficient priority to preempt current execution.

This field resets to an architecturally UNKNOWN value.

**State, bits [29:28]**

The state of the interrupt. This field has one of the following values:

<table>
<thead>
<tr>
<th>State</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Inactive</td>
</tr>
<tr>
<td>0b01</td>
<td>Pending</td>
</tr>
<tr>
<td>0b10</td>
<td>Active</td>
</tr>
<tr>
<td>0b11</td>
<td>Active and pending</td>
</tr>
</tbody>
</table>

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the inactive state are ignored, except for the purpose of generating virtual maintenance interrupts.

Note

For hardware interrupts, the active and pending state is held in the Distributor rather than the virtual CPU interface. A hypervisor must only use the active and pending state for software originated interrupts, which are typically associated with virtual devices, or for SGIs.

This field resets to an architecturally UNKNOWN value.

**Priority, bits [27:23]**

The priority of this interrupt.

This field resets to an architecturally UNKNOWN value.

**Bits [22:20]**

Reserved, RES0.

**pINTID, bits [19:10]**

The function of this field depends on the value of GICH_LR<n>.HW.

When GICH_LR<n>.HW == 0:

- Bit [19] indicates whether the interrupt triggers an EOI maintenance interrupt. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits [18:13] are reserved, SBZ.
- If the vINTID field value corresponds to an SGI (that is, 0-15), bits [12:10] contain the number of the requesting PE. This appears in the corresponding field of GICV_IAR or GICV_AIAR. If the vINTID field value is not 0-15, this field must be cleared to 0.

When GICH_LR<n>.HW == 1:

- This field indicates the pINTID that the hypervisor forwards to the Distributor. This field is only required to implement enough bits to hold a valid value for the ID configuration. Any unused higher order bits are RAZ/WI.
- If the value of pINTID is 0-15 or 1020-1023, behavior is UNPREDICTABLE. If the value of pINTID is 16-31, this field applies to the PPI associated with this same PE as the virtual CPU interface requesting the deactivation.

This field resets to an architecturally UNKNOWN value.
vINTID, bits [9:0]

This INTID is returned to the VM when the interrupt is acknowledged through GICV_IAR. Each valid interrupt stored in the List registers must have a unique vINTID for that virtual CPU interface. If the value of vINTID is 1020-1023, behavior is UNPREDICTABLE.

This field resets to an architecturally UNKNOWN value.

Accessing the GICH_LR<n>

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICH_LR<n> provides equivalent functionality.
- For AArch64 implementations, ICH_LR<n>_EL2 provides equivalent functionality.

**GICH_LR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0100 + 4n</td>
<td>GICH_LR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICH_MISR characteristics are:

**Purpose**

Indicates which maintenance interrupts are asserted.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICH_MISR is a 32-bit register.

**Field descriptions**

The GICH_MISR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>VGrp1D</td>
</tr>
<tr>
<td>29</td>
<td>VGrp1E</td>
</tr>
<tr>
<td>28</td>
<td>VGrp0D</td>
</tr>
<tr>
<td>27</td>
<td>VGrp0E</td>
</tr>
<tr>
<td>26</td>
<td>NP</td>
</tr>
<tr>
<td>25</td>
<td>LRENP</td>
</tr>
<tr>
<td>24</td>
<td>U</td>
</tr>
<tr>
<td>23</td>
<td>EOI</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**VGrp1D, bit [7]**

vPE Group 1 Disabled.

<table>
<thead>
<tr>
<th>VGrp1D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 1 Disabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 1 Disabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when GICH_HCR.VGrp1DIE == 1 and GICH_VMCR.VENG1 == 0.

This field resets to 0.

**VGrp1E, bit [6]**

vPE Group 1 Enabled.

<table>
<thead>
<tr>
<th>VGrp1E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>vPE Group 1 Enabled maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>vPE Group 1 Enabled maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when GICH_HCR.VGrp1EIE == 1 and GICH_VMCR.VENG1 == 1.

This field resets to 0.

**VGrp0D, bit [5]**

vPE Group 0 Disabled.
VGrp0D, bit [4]

vPE Group 0 Disabled maintenance interrupt not asserted.

This maintenance interrupt is asserted when GICH_HCR.VGrp0DIE == 1 and GICH_VMCR.VENG0 == 0.

This field resets to 0.

VGrp0E, bit [4]

vPE Group 0 Enabled maintenance interrupt not asserted.

This maintenance interrupt is asserted when GICH_HCR.VGrp0EIE == 1 and GICH_VMCR.VENG0 == 1.

This field resets to 0.

NP, bit [3]

No Pending maintenance interrupt not asserted.

This maintenance interrupt is asserted when GICH_HCR.NPIE == 1 and no List register is in the pending state.

This field resets to 0.

LRENP, bit [2]

List Register Entry Not Present maintenance interrupt not asserted.

This maintenance interrupt is asserted when GICH_HCR.LRENPIE == 1 and GICH_HCR.EOICount is nonzero.

This field resets to 0.

U, bit [1]

Underflow maintenance interrupt not asserted.

This maintenance interrupt is asserted when GICH_HCR.UIE == 1 and zero or one of the List register entries are marked as a valid interrupt.

This field resets to 0.

EOI, bit [0]

End Of Interrupt.
### EOI

<table>
<thead>
<tr>
<th>EOI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>End Of Interrupt maintenance interrupt not asserted.</td>
</tr>
<tr>
<td>0b1</td>
<td>End Of Interrupt maintenance interrupt asserted.</td>
</tr>
</tbody>
</table>

This maintenance interrupt is asserted when at least one bit in `GICH_EISR` == 1.

This field resets to 0.

**Note**

A List register is in the pending state only if the corresponding `GICH_LR<n>` value is 0b01, that is, pending. The active and pending state is not included.

### Accessing the GICH_MISR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, `ICH_MISR` provides equivalent functionality.
- For AArch64 implementations, `ICH_MISR_EL2` provides equivalent functionality.

A maintenance interrupt is asserted only if at least one bit is set to 1 in this register and if `GICH_HCR.En` == 1.

**GICH_MISR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0010</td>
<td>GICH_MISR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GICH_VMCR, Virtual Machine Control Register

The GICH_VMCR characteristics are:

**Purpose**

Enables the hypervisor to save and restore the virtual machine view of the GIC state. This register is updated when a virtual machine updates the virtual CPU interface registers.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICH_VMCR is a 32-bit register.

**Field descriptions**

The GICH_VMCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Bit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>VPMR</td>
<td>Virtual priority mask. The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE. This alias field is updated when a VM updates GICV_PMR.Priority. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>23-21</td>
<td>VBPR0</td>
<td>Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 1. This alias field is updated when a VM updates GICV_BPR.Binary_Point. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>20-18</td>
<td>VBPR1</td>
<td>Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 0. This alias field is updated when a VM updates GICV_ABPR.Binary_Point. This field resets to an architecturally UNKNOWN value.</td>
</tr>
<tr>
<td>17-10</td>
<td>RES0</td>
<td>Reserved. RES0.</td>
</tr>
<tr>
<td>9-0</td>
<td>VEOIM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VCBPR</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VFIQEn</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VAckCtl</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VENG0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VENG1</td>
<td></td>
</tr>
</tbody>
</table>

VPMR, bits [31:24]

Virtual priority mask. The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

This alias field is updated when a VM updates GICV_PMR.Priority.

This field resets to an architecturally UNKNOWN value.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 1.

This alias field is updated when a VM updates GICV_BPR.Binary_Point.

This field resets to an architecturally UNKNOWN value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 0.

This alias field is updated when a VM updates GICV_ABPR.Binary_Point.

This field resets to an architecturally UNKNOWN value.

Bits [17:10]

Reserved, RES0.
**VEOIM, bit [9]**

Virtual EOI mode. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VEOIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A write of an INTID to GICV_EOIR or GICV_AEOIR drops the priority of the interrupt with that INTID, and also deactivates that interrupt.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write of an INTID to GICV_EOIR or GICV_AEOIR only drops the priority of the interrupt with that INTID. Software must write to GICV_DIR to deactivate the interrupt.</td>
</tr>
</tbody>
</table>

This alias field is updated when a VM updates GICV_CTLR.EOImode.

This field resets to an architecturally UNKNOWN value.

**Bits [8:5]**

Reserved, RES0.

**VCBPR, bit [4]**

Virtual Common Binary Point Register. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VCBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICV_ABPR determines the preemption group for Group 1 interrupts.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICV_BPR determines the preemption group for Group 1 interrupts.</td>
</tr>
</tbody>
</table>

This alias field is updated when a VM updates GICV_CTLR.CBPR.

This field resets to an architecturally UNKNOWN value.

**VFIQEn, bit [3]**

Virtual FIQ enable. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VFIQEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 virtual interrupts are presented as virtual IRQs.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 virtual interrupts are presented as virtual FIQs.</td>
</tr>
</tbody>
</table>

This alias field is updated when a VM updates GICV_CTLR.FIQEn.

This field resets to an architecturally UNKNOWN value.

**VAckCtl, bit [2]**

Virtual AckCtl. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VAckCtl</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt.</td>
</tr>
</tbody>
</table>

This alias field is updated when a VM updates GICV_CTLR.AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

This field resets to an architecturally UNKNOWN value.
**VENG1, bit [1]**

Virtual interrupt enable, Group 1. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VENG1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 1 virtual interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 1 virtual interrupts are enabled.</td>
</tr>
</tbody>
</table>

This alias field is updated when a VM updates `GICV_CTLR.EnableGrp1`.

This field resets to an architecturally `UNKNOWN` value.

**VENG0, bit [0]**

Virtual interrupt enable, Group 0. Possible values of this bit are:

<table>
<thead>
<tr>
<th>VENG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 virtual interrupts are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 virtual interrupts are enabled.</td>
</tr>
</tbody>
</table>

This alias field is updated when a VM updates `GICV_CTLR.EnableGrp0`.

This field resets to an architecturally `UNKNOWN` value.

---

**Note**

A List register is in the pending state only if the corresponding `GICH_LR<n>` value is `0b01`, that is, pending. The active and pending state is not included.

**Accessing the GICH_VMCR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, `ICH_VMCR` provides equivalent functionality.
- For AArch64 implementations, `ICH_VMCR_EL2` provides equivalent functionality.

**GICH_VMCR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0008</td>
<td>GICH_VMCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RW**.
- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.
GICH_VTR, Virtual Type Register

The GICH_VTR characteristics are:

**Purpose**

Indicates the number of implemented virtual priority bits and List registers.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICH_VTR is a 32-bit register.

**Field descriptions**

The GICH_VTR bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>PRibits</th>
<th>PREbits</th>
<th>IDbits</th>
<th>SEIS</th>
<th>A3V</th>
<th>RES0</th>
<th>ListRegs</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td>3</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td>4</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td>5</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td>6</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td>7</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td>8</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td>9</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td>10</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td>11</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td>12</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td>13</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td>14</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td>15</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td>16</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td>17</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td>18</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td>19</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td>20</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>21</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>22</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td>23</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td>24</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>25</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td>26</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td>27</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>28</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>29</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>30</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td>31</td>
<td>0</td>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**PRIbits, bits [31:29]**

The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

**PREbits, bits [28:26]**

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of GICH_VTR.PRIbits.

**IDbits, bits [25:23]**

The number of virtual interrupt identifier bits supported:

<table>
<thead>
<tr>
<th>IDbits</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>16 bits.</td>
</tr>
<tr>
<td>001</td>
<td>24 bits.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**SEIS, bit [22]**

SEI support. Indicates whether the virtual CPU interface supports generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The virtual CPU interface logic does not support generation of SEIs.</td>
</tr>
<tr>
<td>1</td>
<td>The virtual CPU interface logic supports generation of SEIs.</td>
</tr>
</tbody>
</table>
A3V, bit [21]

Affinity 3 valid. Possible values are:

<table>
<thead>
<tr>
<th>A3V</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual CPU interface logic only supports zero values of the Aff3 field in <code>ICC_SGI0R_EL1</code>, <code>ICC_SGI1R_EL1</code>, and <code>ICC_ASGI1R_EL1</code>.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual CPU interface logic supports nonzero values of the Aff3 field in <code>ICC_SGI0R_EL1</code>, <code>ICC_SGI1R_EL1</code>, and <code>ICC_ASGI1R_EL1</code>.</td>
</tr>
</tbody>
</table>

Bits [20:5]

Reserved, RES0.

ListRegs, bits [4:0]

The number of implemented List registers, minus one.

Accessing the GICH_VTR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, `ICH_VTR` provides equivalent functionality.
- For AArch64 implementations, `ICH_VTR_EL2` provides equivalent functionality.

GICH_VTR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual interface control</td>
<td>0x0004</td>
<td>GICH_VTR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RO**.
- When `IsAccessSecure()` accesses to this register are **RO**.
- When `!IsAccessSecure()` accesses to this register are **RO**.
GICR_CLRLPIR, Clear LPI Pending Register

The GICR_CLRLPIR characteristics are:

**Purpose**

Clears the pending state of the specified LPI.

**Configuration**

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_CLRLPIR is a 64-bit register.

**Field descriptions**

The GICR_CLRLPIR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>pINTID</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**pINTID, bits [31:0]**

The INTID of the physical LPI.

**Note**

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER.Idbits field. Unimplemented bits are RES0.

**Accessing the GICR_CLRLPIR**

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality of this register is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if any of the following apply:

- GICR_CTLR.EnableLPIs == 0.
- The pINTID value specifies an unimplemented LPI.
- The pINTID value specifies an LPI that is not pending.

**GICR_CLRLPIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
GICR_CTLR, Redistributor Control Register

The GICR_CTLR characteristics are:

**Purpose**

Controls the operation of a Redistributor, and enables the signaling of LPIs by the Redistributor to the connected PE.

**Configuration**

Some or all RW fields of this register have defined reset values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_CTLR is a 32-bit register.

**Field descriptions**

The GICR_CTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>UWP</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>DPG1S</td>
</tr>
<tr>
<td>28</td>
<td>DPG1NS</td>
</tr>
<tr>
<td>27</td>
<td>DPG0</td>
</tr>
<tr>
<td>26</td>
<td>RES0</td>
</tr>
<tr>
<td>25</td>
<td>RWP</td>
</tr>
<tr>
<td>24</td>
<td>RES0</td>
</tr>
<tr>
<td>23</td>
<td>CES</td>
</tr>
<tr>
<td>22</td>
<td>EnableLPIs</td>
</tr>
</tbody>
</table>

**UWP, bit [31]**

Upstream Write Pending. Read-only. Indicates whether all upstream writes have been communicated to the Distributor.

<table>
<thead>
<tr>
<th>UWP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The effects of all upstream writes have been communicated to the Distributor, including any Generate SGI packets.</td>
</tr>
<tr>
<td>1</td>
<td>Not all the effects of upstream writes, including any Generate SGI packets, have been communicated to the Distributor.</td>
</tr>
</tbody>
</table>

**Bits [30:27]**

Reserved, RES0.

**DPG1S, bit [26]**

Disable Processor selection for Group 1 Secure interrupts. When GICR_TYPER.DPGS == 1:

<table>
<thead>
<tr>
<th>DPG1S</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A Group 1 Secure SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Secure Group 1 interrupts are enabled.</td>
</tr>
<tr>
<td>1</td>
<td>A Group 1 Secure SPI configured to use the 1 of N distribution model cannot select this PE.</td>
</tr>
</tbody>
</table>

When GICR_TYPER.DPGS == 0 this bit is RAZ/WI.

When GICD_CTLR.DS==1, this field is RAZ/WI. In GIC implementations that support two Security states, this field is only accessible by Secure accesses, and is RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether this bits affect the selection of PEs for interrupts using the 1 of N distribution model when GICD_CTLR.ARE_S==0.
This field resets to 0.

DPG1NS, bit [25]

Disable Processor selection for Group 1 Non-secure interrupts. When GICR_TYPER.DPGS == 1:

<table>
<thead>
<tr>
<th>DPG1NS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A Group 1 Non-secure SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Non-secure Group 1 interrupts are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Group 1 Non-secure SPI configured to use the 1 of N distribution model cannot select this PE.</td>
</tr>
</tbody>
</table>

When GICR_TYPER.DPGS == 0 this bit is RAZ/WI.

It is IMPLEMENTATION DEFINED whether this bits affect the selection of PEs for interrupts using the 1 of N distribution model when GICD_CTLR.ARE_NS==0.

This field resets to 0.

DPG0, bit [24]

Disable Processor selection for Group 0 interrupts. When GICR_TYPER.DPGS == 1:

<table>
<thead>
<tr>
<th>DPG0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A Group 0 SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Group 0 interrupts are enabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Group 0 SPI configured to use the 1 of N distribution model cannot select this PE.</td>
</tr>
</tbody>
</table>

When GICR_TYPER.DPGS == 0 this bit is RAZ/WI.

When GICD_CTLR.DS==1, this field is always accessible. In GIC implementations that support two Security states, this field is RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether this bits affect the selection of PEs for interrupts using the 1 of N distribution model when GICD_CTLR.ARE_S==0.

This field resets to 0.

Bits [23:4]

Reserved, RES0.

RWP, bit [3]

Register Write Pending. This bit indicates whether a register write for the current Security state is in progress or not.

<table>
<thead>
<tr>
<th>RWP</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 0b0   | The effect of all previous writes to the following registers are visible to all agents in the system:  
  • GICR_ICENABLER0  
  • GICR_CTLR DPG1S  
  • GICR_CTLR DPG1NS  
  • GICR_CTLR DPG0  
  • GICR_CTLR, which clears EnableLPIs from 1 to 0. |
| 0b1   | The effect of all previous writes to the following registers are not guaranteed by the architecture to be visible yet to the all agents in the system as the changes are still being propagated:  
  • GICR_ICENABLER0  
  • GICR_CTLR DPG1S  
  • GICR_CTLR DPG1NS  
  • GICR_CTLR DPG0  
  • GICR_CTLR, which clears EnableLPIs from 1 to 0. |


Bit [2]

Reserved, RES0.

CES, bit [1]

Clear Enable Supported.

This bit is read-only.

<table>
<thead>
<tr>
<th>CES</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The IRI does not indicate whether GICR_CTLR.EnableLPIs is RES1 once set.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICR_CTLR.EnableLPIs is not RES1 once set.</td>
</tr>
</tbody>
</table>

Implementing GICR_CTLR.EnableLPIs as programmable and not reporting CES == 1 is deprecated.

Implementing GICR_CTLR.EnableLPIs as RES1 once set is deprecated.

When GICR_CLTR.CES == 0, software cannot assume that GICR_CTLR.EnableLPIs is programmable without observing the bit being cleared.

EnableLPIs, bit [0]

In implementations where affinity routing is enabled for the Security state:

<table>
<thead>
<tr>
<th>EnableLPIs</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>LPI support is disabled. Any doorbell interrupt generated as a result of a write to a virtual LPI register must be discarded, and any ITS translation requests or commands involving LPIs in this Redistributor are ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>LPI support is enabled.</td>
</tr>
</tbody>
</table>

Note

If GICR_TYPER.LPIS == 0, this field is RES0. If GICD_CTLR.ARE_NS is written from 1 to 0 when this bit is 1, behavior is an IMPLEMENTATION DEFINED choice between clearing GICR_CTLR.EnableLPIs to 0 or maintaining its current value.

When affinity routing is not enabled for the Non-secure state, this bit is RES0.

When written from 0 to 1, the Redistributor loads the LPI Pending table from memory to check for any pending interrupts.

After it has been written to 1, it is IMPLEMENTATION DEFINED whether the bit becomes RES1 or can be cleared by to 0.

Where the bit remains programmable:

- Software must observe GICR_CTLR.RWP==0 after clearing GICR_CTLR.EnableLPIs from 1 to 0 before writing GICR_PENDBASER or GICR_PROPBASER, otherwise behavior is UNPREDICTABLE.
- Software must observe GICR_CTLR.RWP==0 after clearing GICR_CTLR.EnableLPIs from 1 to 0 before setting GICR_CTLR.EnableLPIs to 1, otherwise behavior is UNPREDICTABLE.

Note

If one or more ITS is implemented, Arm strongly recommends that all LPIs are mapped to another Redistributor before GICR_CTLR.EnableLPIs is cleared to 0.

This field resets to 0.

The participation of a PE in the 1 of N distribution model for a given interrupt group is governed by the concatenation of GICR_WAKER.ProcessorSleep, the appropriate GICR_CTLR.DPG{1, 0} bit, and the PE interrupt group enable. The behavior options are:
The PE cannot be selected.

The PE can be selected.

The PE cannot be selected.

The PE cannot be selected when \texttt{GICD_CTLR.E1NWF} == 0. When \texttt{GICD_CTLR.E1NWF} == 1, the mechanism by which PEs are selected is \texttt{IMPLEMENTATION DEFINED}.

If an SPI using the 1 of N distribution model has been forwarded to the PE and a write to \texttt{GICR_CTLR} occurs that changes the DPG bit for the interrupt group of the SPI, the IRI must attempt to select a different target PE for the SPI. This might have no effect on the forwarded SPI if it has already been activated.

**Accessing the GICR_CTLR**

\texttt{GICR_CTLR} can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0000</td>
<td>GICR_CTLR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{GICD_CTLR.DS} == 0b0 accesses to this register are \texttt{RW}.
- When \texttt{IsAccessSecure()} accesses to this register are \texttt{RW}.
- When \texttt{!IsAccessSecure()} accesses to this register are \texttt{RW}.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICR_ICACTIVER0, Interrupt Clear-Active Register 0

The GICR_ICACTIVER0 characteristics are:

Purpose

Deactivates the corresponding SGI or PPI. These registers are used when saving and restoring GIC state.

Configuration

RW fields in this register reset to architecturally **UNKNOWN** values.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICACTIVER0 is a 32-bit register.

Field descriptions

The GICR_ICACTIVER0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Clear_active_bit<x>, bit [x], for x = 0 to 31**

Removes the active state from interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th><strong>Clear_active_bit&lt;x&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Accessing the GICR_ICACTIVER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICACTIVER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by **GICD_ICACTIVER<n>** with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by **GICD_ICACTIVER<n>**.

When **GICD_CTLR.DS == 0**, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

**GICR_ICACTIVER0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
</table>

Page 3718
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_ICACTIVER\(<n>\)E characteristics are:

**Purpose**

Removes the active state from the corresponding PPI.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when GIC, \(\geq 3.1\) is implemented. Otherwise, direct accesses to GICR_ICACTIVER\(<n>\)E are **RES0**.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICACTIVER\(<n>\)E is a 32-bit register.

**Field descriptions**

The GICR_ICACTIVER\(<n>\)E bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignments</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
<td>Clear_active_bit(&lt;x&gt;), bit [x], for x = 0 to 31</td>
</tr>
</tbody>
</table>

**Clear_active_bit\(<x>\), bit [x], for x = 0 to 31**

For the extended PPIs, removes the active state to interrupt number \(x\). Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_active_bit(&lt;x&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

For INTID \(m\), when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICACTIVER\(<n>\)E number, \(n\), is given by \(n = (m - 1024) \div 32\).
- The offset of the required GICR_ICACTIVER\(<n>\)E is \((0x200 + (4*n))\).
- The bit number of the required group modifier bit in this register is \((m - 1024) \mod 32\).

**Accessing the GICR_ICACTIVER\(<n>\)E**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICACTIVER\(<n>\)E, the corresponding bit is **RES0**.
When \texttt{GICD_CTLR.DS==0}, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

\textbf{GICR\_ICACTIVER<}\textit{n>E can be accessed through the memory-mapped interfaces:}

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0380</td>
<td>GICR_ICACTIVER&lt;\textit{n}&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{GICD\_CTRL.DS == 0b0} accesses to this register are \textbf{RW}.
- When \texttt{IsAccessSecure()} accesses to this register are \textbf{RW}.
- When \texttt{!IsAccessSecure()} accesses to this register are \textbf{RW}.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**GICR_ICENABLER0, Interrupt Clear-Enable Register 0**

The GICR_ICENABLER0 characteristics are:

**Purpose**

Disables forwarding of the corresponding SGI or PPI to the CPU interfaces.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICENABLER0 is a 32-bit register.

**Field descriptions**

The GICR_ICENABLER0 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear_enable_bit&lt;x&gt;, bit [x], for x = 0 to 31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Clear_enable_bit<x>, bit [x], for x = 0 to 31**

For PPIs and SGIs, controls the forwarding of interrupt number x to the CPU interfaces. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_enable_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the GICR_ICENABLER0**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICENABLER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ICENABLER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ICENABLER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

**GICR_ICENABLER0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
</table>

Page 3722
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_ICENABLER<n>E characteristics are:

**Purpose**

Disables forwarding of the corresponding PPI to the CPU interfaces.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_ICENABLER<n>E are RES0.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICENABLER<n>E is a 32-bit register.

**Field descriptions**

The GICR_ICENABLER<n>E bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Clear_enable_bit<x>, bit [x], for x = 0 to 31 |

**Clear_enable_bit<x>, bit [x], for x = 0 to 31**

For the extended PPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_enable_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICENABLER<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ICENABLER<n>E is (0x180 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

**Accessing the GICR_ICENABLER<n>E**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICENABLER<n>E, the corresponding bit is RES0.
When `GICD_CTLR.DS==0`, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICR_ICENABLER<n>E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC</td>
<td>SGI_base</td>
<td>0x0180</td>
<td>GICR_ICENABLER&lt;n&gt;E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ 4n</td>
<td></td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RW**.
- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.
GICR_ICFGR0, Interrupt Configuration Register 0

The GICR_ICFGR0 characteristics are:

**Purpose**

Determines whether the corresponding SGI is edge-triggered or level-sensitive.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICFGR0 is a 32-bit register.

**Field descriptions**

The GICR_ICFGR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Int_config&lt;x&gt;</strong>, bits [2x+1:2x], for x = 0 to 15</td>
<td>Indicates whether the interrupt with ID 16n + x is level-sensitive or edge-triggered.</td>
</tr>
</tbody>
</table>

**Int_config<x>**, bits [2x+1:2x], for x = 0 to 15

Indicates whether the interrupt with ID 16n + x is level-sensitive or edge-triggered.

**Int_config[0]** (bit [2x]) is **RES0**.

Possible values of **Int_config[1]** (bit [2x+1]) are:

<table>
<thead>
<tr>
<th><strong>Int_config&lt;1&gt;</strong></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Corresponding interrupt is level-sensitive.</td>
</tr>
<tr>
<td>0b01</td>
<td>Corresponding interrupt is edge-triggered.</td>
</tr>
</tbody>
</table>

For SGIs, **Int_config[1]** is RAO/WI.

A read of this bit always returns the correct value to indicate the interrupt triggering method.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the GICR_ICFGR0**

This register is used when affinity routing is enabled.

When affinity routing is disabled for the Security state of an interrupt, the field for that interrupt is **RES0** and an implementation is permitted to make the field RAZ/WI in this case. Equivalent functionality is provided by GICD_ICFGR<n> with n=0.

When **GICD_CTLR.DS==0**, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

**GICR_ICFGR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
</table>
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICR_ICFGR1, Interrupt Configuration Register 1

The GICR_ICFGR1 characteristics are:

**Purpose**

Determines whether the corresponding PPI is edge-triggered or level-sensitive.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

A copy of this register is provided for each Redistributor.

For each supported PPI, it is **IMPLEMENTATION DEFINED** whether software can program the corresponding Int_config field.

Changing Int_config when the interrupt is individually enabled is **UNPREDICTABLE**.

Changing the interrupt configuration between level-sensitive and edge-triggered (in either direction) at a time when there is a pending interrupt will leave the interrupt in an **UNKNOWN** pending state.

**Attributes**

GICR_ICFGR1 is a 32-bit register.

**Field descriptions**

The GICR_ICFGR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td><strong>Int_config&lt;x&gt;, bits [2x+1:2x], for x = 0 to 15</strong></td>
</tr>
<tr>
<td>30</td>
<td><strong>Int_config[x], bits [2x+1:2x], for x = 0 to 15</strong></td>
</tr>
</tbody>
</table>

Indicates whether the interrupt with ID 16n + x is level-sensitive or edge-triggered.

Int_config[0] (bit [2x]) is **RES0**.

Possible values of Int_config[1] (bit [2x+1]) are:

<table>
<thead>
<tr>
<th><strong>Int_config&lt;x&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Corresponding interrupt is level-sensitive.</td>
</tr>
<tr>
<td>0b01</td>
<td>Corresponding interrupt is edge-triggered.</td>
</tr>
</tbody>
</table>

A read of this bit always returns the correct value to indicate the interrupt triggering method.

For PPIs, Int_config[1] is programmable unless the implementation supports two Security states and the bit corresponds to a Group 0 or Secure Group 1 interrupt, in which case the bit is RAZ/WI to Non-secure accesses.

This field resets to an architecturally **UNKNOWN** value.

**Accessing the GICR_ICFGR1**

This register is used when affinity routing is enabled.

When affinity routing is disabled for the Security state of an interrupt, the field for that interrupt is **RES0** and an implementation is permitted to make the field RAZ/WI in this case. Equivalent functionality is provided by GICD_ICFGR<n> with n=1.
GICR_ICFGR1 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0xC04</td>
<td>GICR_ICFGR1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_ICFGR<n>E characteristics are:

**Purpose**

Determines whether the corresponding PPI in the extended PPI range is edge-triggered or level-sensitive.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_ICFGR<n>E are RES0.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICFGR<n>E is a 32-bit register.

**Field descriptions**

The GICR_ICFGR<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Int_config&lt;x&gt;, bit [x], for x = 0 to 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

Indicates whether the interrupt with ID 16n + x is level-sensitive or edge-triggered.

Int_config[0] (bit [2x]) is RES0.

Possible values of Int_config[1] (bit [2x+1]) are:

<table>
<thead>
<tr>
<th>Int_config&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The corresponding interrupt is level-sensitive.</td>
</tr>
<tr>
<td>0b1</td>
<td>The corresponding interrupt is edge-triggered.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

For each supported extended PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

**Accessing the GICR_ICFGR<n>E**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICFGR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.
GICR_ICFGR\(n\)E can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0C00 + 4n</td>
<td>GICR_ICFGR(n)E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICR_ICPENDR0, Interrupt Clear-Pending Register 0

The GICR_ICPENDR0 characteristics are:

**Purpose**

Removes the pending state from the corresponding SGI or PPI.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICPENDR0 is a 32-bit register.

**Field descriptions**

The GICR_ICPENDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Clear_pending_bit&lt;x&gt;, bit [x], for x = 0 to 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Clear_pending_bit<x>, bit [x], for x = 0 to 31

Removes the pending state from interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_pending_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending. If written, has no effect.</td>
</tr>
</tbody>
</table>
| 0b1                  | If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases:  
  - If the interrupt is not pending and is not active and pending.  
  - If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending. |

This field resets to an architecturally **UNKNOWN** value.

**Accessing the GICR_ICPENDR0**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICPENDR0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ICPENDR<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ICENABLER<n>. 
When `GICD_CTLR.DS == 0`, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

**GICR_ICPENDR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0280</td>
<td>GICR_ICPENDR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RW**.
- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.
The GICR_ICPENDR\(\langle n\rangle\)E characteristics are:

**Purpose**

Removes the pending state from the corresponding PPI.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when GIC, \(\geq 3.1\) is implemented. Otherwise, direct accesses to GICR_ICPENDR\(\langle n\rangle\)E are RES0.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ICPENDR\(\langle n\rangle\)E is a 32-bit register.

**Field descriptions**

The GICR_ICPENDR\(\langle n\rangle\)E bit assignments are:

| Clear_pending_bit\(<x>\), bit \([x]\), for \(x = 0\) to \(31\) |
|---------------------|-----------------------------|
| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |

For the extended PPIs, removes the pending state to interrupt number \(x\). Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Clear_pending_bit(&lt;x&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is not pending and is not active and pending.</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICR_ISPENDR(\langle n\rangle)E. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

For INTID \(m\), when DIV and MOD are the integer division and modulo operations:
The corresponding GICR_ICPENDR\(n\)E number, \(n\), is given by \(n = (m-1024) \div 32\).
- The offset of the required GICR_ICPENDR\(n\)E is \((0x200 + (4*n))\).
- The bit number of the required group modifier bit in this register is \((m-1024) \mod 32\).

## Accessing the GICR_ICPENDR\(n\)E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICPENDR\(n\)E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

\[\text{GICR_ICPENDR}\(n\)E can be accessed through the memory-mapped interfaces:}\]

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0280 + 4n</td>
<td>GICR_ICPENDR(n)E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICR_IGROUPR0 characteristics are:

**Purpose**

Controls whether the corresponding SGI or PPI is in Group 0 or Group 1.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is available in all GIC configurations. If the GIC implementation supports two Security states, this register is Secure.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_IGROUPR0 is a 32-bit register.

**Field descriptions**

The GICR_IGROUPR0 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Redistributor_group_status_bit&lt;x&gt;, bit [x], for x = 0 to 31</strong></td>
<td><strong>Group status bit. In this register:</strong></td>
</tr>
<tr>
<td>0b0</td>
<td>Bits [31:16] are group status bits for PPIs.</td>
</tr>
<tr>
<td></td>
<td>Bits [15:0] are group status bits for SGIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>When <strong>GICD_CTLR.DS==1</strong>, the corresponding interrupt is Group 0.</td>
</tr>
<tr>
<td></td>
<td>When <strong>GICD_CTLR.DS==0</strong>, the corresponding interrupt is Secure.</td>
</tr>
<tr>
<td></td>
<td>When <strong>GICD_CTLR.DS==1</strong>, the corresponding interrupt is Group 1.</td>
</tr>
<tr>
<td></td>
<td>When <strong>GICD_CTLR.DS==0</strong>, the corresponding interrupt is Non-secure Group 1.</td>
</tr>
</tbody>
</table>

When **GICD_CTLR.DS == 0**, the bit that corresponds to the interrupt is concatenated with the equivalent bit in **GICR_IGRPMODR0** to form a 2-bit field that defines an interrupt group. The encoding of this field is at **GICR_IGRPMODR0**.

This field resets to an architecturally **UNKNOWN** value.

The considerations for the reset value of this register are the same as those for **GICD_IGROUP<n>** with n=0.
Accessing the GICR_IGROUPR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGROUPR0, the corresponding bit is RES0 and equivalent functionality is provided by GICD_IGROUPR<n> with n=0.

When GICD_CTLR_DS == 0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

---

**Note**

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

---

**GICR_IGROUPR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0080</td>
<td>GICR_IGROUPR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR_DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_IGROUPR<n>E characteristics are:

**Purpose**

Controls whether the corresponding PPI is in Group 0 or Group 1.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_IGROUPR<n>E are RES0.

When GICD_CLTR.DS==0, this register is Secure.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_IGROUPR<n>E is a 32-bit register.

**Field descriptions**

The GICR_IGROUPR<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group_status_bit&lt;x&gt;, bit [x], for x = 0 to 31</td>
<td>When GICD_CTLR.DS==1, the corresponding interrupt is Group 0.</td>
</tr>
<tr>
<td></td>
<td>When GICD_CTLR.DS==0, the corresponding interrupt is Secure.</td>
</tr>
<tr>
<td>0b1</td>
<td>When GICD_CTLR.DS==1, the corresponding interrupt is Group 1.</td>
</tr>
<tr>
<td></td>
<td>When GICD_CTLR.DS==0, the corresponding interrupt is Non-secure Group 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICR_IGRPMODR<n>E to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICR_IGRPMODR<n>E.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IGROUPR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_IGROUPR<n>E is (0x800 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.
Accessing the GICR_IGROUPR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGROUPR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_IGROUPR<n>E can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0080 + 4n</td>
<td>GICR_IGROUPR&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
**GICR_IGRPMODR0, Interrupt Group Modifier Register 0**

The GICR_IGRPMODR0 characteristics are:

**Purpose**

When $\text{GICD_CTLR.DS}==0$, this register together with the $\text{GICR_IGROUPR0}$ register, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

When GICD_CLTR.DS==0, this register is Secure.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_IGRPMODR0 is a 32-bit register.

**Field descriptions**

The GICR_IGRPMODR0 bit assignments are:

<table>
<thead>
<tr>
<th>Group modifier bit</th>
<th>Group status bit</th>
<th>Definition</th>
<th>Short name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>0b0</td>
<td>Secure Group 0</td>
<td>G0S</td>
</tr>
<tr>
<td>0b0</td>
<td>0b1</td>
<td>Non-secure Group 1</td>
<td>G1NS</td>
</tr>
<tr>
<td>0b1</td>
<td>0b0</td>
<td>Secure Group 1</td>
<td>G1S</td>
</tr>
<tr>
<td>0b1</td>
<td>0b1</td>
<td>Reserved, treated as Non-secure Group 1</td>
<td>-</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the GICR_IGRPMODR0**

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGRPMODR0, the corresponding bit is RES0 and equivalent functionality is provided by $\text{GICD_IGRPMODR<n>}$ with $n=0$.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by $\text{GICD_IGRPMODR<n>}$.

When $\text{GICD_CTLR.ARE_S} == 0$ or $\text{GICD_CTLR.DS} == 1$, GICR_IGRPMODR0 is RES0. An implementation can make this register RAZ/WI in this case.
When `GICD_CTLR.DS==0`, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

---

**Note**

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

---

**GICR_IGRPMODR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0D00</td>
<td>GICR_IGRPMODR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.

---

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICR_IGRPMODR<n>E, Interrupt Group Modifier Registers, n = 1 - 2

The GICR_IGRPMODR<n>E characteristics are:

**Purpose**

When GICD_CTLR.DS==0, this register together with the GICR_IGROUPR<n>E registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_IGRPMODR<n>E are RES0.

When GICD_CLTR.DS==0, this register is Secure.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_IGRPMODR<n>E is a 32-bit register.

**Field descriptions**

The GICR_IGRPMODR<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Group modifier bit&lt;x&gt;, bit [x], for x = 0 to 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Group Modifier Bit</td>
</tr>
<tr>
<td>30</td>
<td>Group Modifier Bit</td>
</tr>
<tr>
<td>29</td>
<td>Group Modifier Bit</td>
</tr>
<tr>
<td>28</td>
<td>Group Modifier Bit</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IGRPMODR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_IGRPMODR<n>E is (0x000 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.
Accessing the GICR_IGRPMODR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGRPMODR<n>E, the corresponding bit is Res0.

When GICD_CTLR.DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICR_IGRPMODR<n>E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0D00 + 4n</td>
<td>GICR_IGRPMODR&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_IIDR characteristics are:

**Purpose**

Provides information about the implementer and revision of the Redistributor.

**Configuration**

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

**Attributes**

GICR_IIDR is a 32-bit register.

**Field descriptions**

The GICR_IIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>ProductID</td>
</tr>
<tr>
<td>23-20</td>
<td>Reserved</td>
</tr>
<tr>
<td>19-16</td>
<td>Variant</td>
</tr>
<tr>
<td>15-12</td>
<td>Revision</td>
</tr>
<tr>
<td>11-0</td>
<td>Implementer</td>
</tr>
</tbody>
</table>

**ProductID, bits [31:24]**

An implementation-defined product identifier.

**Bits [23:20]**

Reserved, RES0.

**Variant, bits [19:16]**

An implementation-defined variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

**Revision, bits [15:12]**

An implementation-defined revision number. Typically, this field is used to distinguish minor revisions of a product.

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the Redistributor:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.
Accessing the GICR_IIDR

GICR_IIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0004</td>
<td>GICR_IIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The GICR_INVALLR characteristics are:

**Purpose**

Invalidates any cached configuration data of all physical LPIs, causing the GIC to reload the interrupt configuration from the physical LPI Configuration table at the address specified by GICR_PROPBASER.

**Configuration**

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_INVALLR is a 64-bit register.

**Field descriptions**

The GICR_INVALLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>RES0</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>61</td>
<td>RES0</td>
</tr>
<tr>
<td>60</td>
<td>RES0</td>
</tr>
<tr>
<td>59</td>
<td>0</td>
</tr>
<tr>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>57</td>
<td>0</td>
</tr>
<tr>
<td>56</td>
<td>0</td>
</tr>
<tr>
<td>55</td>
<td>0</td>
</tr>
<tr>
<td>54</td>
<td>0</td>
</tr>
<tr>
<td>53</td>
<td>0</td>
</tr>
<tr>
<td>52</td>
<td>0</td>
</tr>
<tr>
<td>51</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
</tr>
<tr>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td>48</td>
<td>0</td>
</tr>
<tr>
<td>47</td>
<td>0</td>
</tr>
<tr>
<td>46</td>
<td>0</td>
</tr>
<tr>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>44</td>
<td>0</td>
</tr>
<tr>
<td>43</td>
<td>0</td>
</tr>
<tr>
<td>42</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>39</td>
<td>0</td>
</tr>
<tr>
<td>38</td>
<td>0</td>
</tr>
<tr>
<td>37</td>
<td>0</td>
</tr>
<tr>
<td>36</td>
<td>0</td>
</tr>
<tr>
<td>35</td>
<td>0</td>
</tr>
<tr>
<td>34</td>
<td>0</td>
</tr>
<tr>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>32</td>
<td>0</td>
</tr>
<tr>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>23</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>0</td>
</tr>
<tr>
<td>20</td>
<td>0</td>
</tr>
<tr>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Bits [63:0]**

Reserved, RES0.

**Note**

If any LPI has been forwarded to the PE and a valid write to GICR_INVALLR is received, the Redistributor must ensure it reloads its properties from memory. This has no effect on the forwarded LPI if it has already been activated.

**Accessing the GICR_INVALLR**

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if no physical LPIs are currently stored in the local Redistributor cache.

**GICR_INVALLR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0080</td>
<td>GICR_INVALLR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are WO.
- When IsAccessSecure() accesses to this register are WO.
- When !IsAccessSecure() accesses to this register are WO.
**GICR_INVLPiR, Redistributor Invalidate LPI Register**

The GICR_INVLPiR characteristics are:

**Purpose**

Invalidates the cached configuration data of a specified LPI, causing the GIC to reload the interrupt configuration from the physical LPI Configuration table at the address specified by `GICR_PROPBASER`.

**Configuration**

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_INVLPiR is a 64-bit register.

**Field descriptions**

The GICR_INVLPiR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>plINTID, bits [31:0]</td>
</tr>
</tbody>
</table>

**Bits [63:32]**

Reserved, RES0.

**plINTID, bits [31:0]**

The INTID of the physical LPI to be cleaned.

**Note**

The size of this field is IMPLEMENTATION DEFINED, and is specified by the `GICD_TYPEPER.IDbits` field. Unimplemented bits are RES0.

**Accessing the GICR_INVLPiR**

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if either:

- The specified LPI is not currently stored in the local Redistributor.
The pINTID field corresponds to an unimplemented LPI.

**GICR_INVLPIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x00A0</td>
<td>GICR_INVLPIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
The GICR_IPRIORITYR\langle n\rangle characteristics are:

**Purpose**

Holds the priority of the corresponding interrupt for each SGI and PPI supported by the GIC.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

A copy of these registers is provided for each Redistributor.

These registers are configured as follows:

- GICR_IPRIORITYR0-GICR_IPRIORITYR3 store the priority of SGIs.
- GICR_IPRIORITYR4-GICR_IPRIORITYR7 store the priority of PPIs.

**Attributes**

GICR_IPRIORITYR\langle n\rangle is a 32-bit register.

**Field descriptions**

The GICR_IPRIORITYR\langle n\rangle bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Priority_offset_3B | Priority_offset_2B | Priority_offset_1B | Priority_offset_0B |

**Priority_offset_3B, bits [31:24]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.

**Priority_offset_2B, bits [23:16]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.

**Priority_offset_1B, bits [15:8]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

This field resets to an architecturally UNKNOWN value.

**Priority_offset_0B, bits [7:0]**

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.
This field resets to an architecturally **UNKNOWN** value.

### Accessing the GICR_IPRIORITYR<n>

These registers are used when affinity routing is enabled for the Security state of the interrupt. When affinity routing is not enabled the bits corresponding to the interrupt are RAZ/WI and **GICD_IPRIORITYR<n>** provides equivalent functionality.

These registers are used for SGIs and PPIs only. Equivalent functionality for SPIs is provided by **GICD_IPRIORITYR<n>**.

These registers are byte-accessible.

When **GICD_CTLR**.DS == 0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

**Note**

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

---

**GICR_IPRIORITYR<n>** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC</td>
<td>SGI_base</td>
<td>0x0400 + 4n</td>
<td>GICR_IPRIORITYR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_IPRIORITYR<n>E characteristics are:

**Purpose**

Holds the priority of the corresponding interrupt for each extended PPI supported by the GIC.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_IPRIORITYR<n>E are **RES0**.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_IPRIORITYR<n>E is a 32-bit register.

**Field descriptions**

The GICR_IPRIORITYR<n>E bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 31  | Priority_offset_3B | Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt. 
|     |               | This field resets to an architecturally **UNKNOWN** value.                    |
| 23  | Priority_offset_2B | Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt. 
|     |               | This field resets to an architecturally **UNKNOWN** value.                    |
| 15  | Priority_offset_1B | Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt. 
|     |               | This field resets to an architecturally **UNKNOWN** value.                    |
| 0   | Priority_offset_0B | Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt. 
|     |               | This field resets to an architecturally **UNKNOWN** value.                    |
For interrupt ID $m$, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR\_IPRIORITY\_{$n$} number, $n$, is given by $n = (m - 1024) \text{ DIV } 4$.
- The offset of the required GICR\_IPRIORITY\_{$n$} register is $(0x400 + (4*n))$.
- The byte offset of the required Priority field in this register is $m \text{ MOD } 4$, where:
  - Byte offset 0 refers to register bits [7:0].
  - Byte offset 1 refers to register bits [15:8].
  - Byte offset 2 refers to register bits [23:16].
  - Byte offset 3 refers to register bits [31:24].

### Accessing the GICR\_IPRIORITY\_{$n$}\_E

When affinity routing is not enabled for the Security state of an interrupt in GICR\_ISACTIVER\_{$n$}\_E, the corresponding bit is \texttt{RES0}.

When $\text{GICD\_CTL}_R.DS==0$:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

Bits corresponding to unimplemented interrupts are RAZ/WI.

---

**Note**

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than once. The effect of the change must be visible in finite time.

---

**GICR\_IPRIORITY\_{$n$}\_E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>$0x400$</td>
<td>GICR_IPRIORITY_{$n$}_E + $4n$</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When $\text{GICD\_CTL}_R.DS == 0b0$ accesses to this register are \texttt{RW}.
- When $\text{IsAccessSecure()}$ accesses to this register are \texttt{RW}.
- When $\text{!IsAccessSecure()}$ accesses to this register are \texttt{RW}.
GICR_ISACTIVER0, Interrupt Set-Active Register 0

The GICR_ISACTIVER0 characteristics are:

**Purpose**

Activates the corresponding SGI or PPI. These registers are used when saving and restoring GIC state.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ISACTIVER0 is a 32-bit register.

**Field descriptions**

The GICR_ISACTIVER0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Set_active_bit<x>, bit [x], for x = 0 to 31

Adds the active state to interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_active_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or is active and pending. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Accessing the GICR_ISACTIVER0**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ISACTIVER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ISACTIVER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISACTIVER0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICR_ISACTIVER<n>E, Interrupt Set-Active Registers, n = 1 - 2

The GICR_ISACTIVER<n>E characteristics are:

**Purpose**

Adds the active state to the corresponding PPI.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_ISACTIVER<n>E are **RES0**.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ISACTIVER<n>E is a 32-bit register.

**Field descriptions**

The GICR_ISACTIVER<n>E bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Set_active_bit&lt;x&gt;, bit [x], for x = 0 to 31</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Set_active_bit<x>, bit [x], for x = 0 to 31**

For the extended PPIs, adds the active state to interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th><strong>Set_active_bit&lt;x&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is active, or active and pending on this PE. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISACTIVER<n>E number, n, is given by \( n = (m-1024) \div 32 \).
- The offset of the required GICR_ISACTIVER<n>E is \((0x200 + (4*n))\).
- The bit number of the required group modifier bit in this register is \((m-1024) \mod 32\).

**Accessing the GICR_ISACTIVER<n>E**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER<n>E, the corresponding bit is **RES0**.
When `GICD_CTLR.DS==0`, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses. Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICR_ISACTIVER<n>E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC</td>
<td>SGI_base</td>
<td>0x0300</td>
<td>GICR_ISACTIVER&lt;n&gt;E</td>
</tr>
<tr>
<td>Redistributor</td>
<td></td>
<td>+ 4n</td>
<td></td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RW**.
- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.
GICR_ISENABLER0, Interrupt Set-Enable Register 0

The GICR_ISENABLER0 characteristics are:

**Purpose**

Enables forwarding of the corresponding SGI or PPI to the CPU interfaces.

**Configuration**

Some or all RW fields of this register have defined reset values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ISENABLER0 is a 32-bit register.

**Field descriptions**

The GICR_ISENABLER0 bit assignments are:

<table>
<thead>
<tr>
<th>Set_enable_bit&lt;x&gt;, bit [x], for x = 0 to 31</th>
</tr>
</thead>
<tbody>
<tr>
<td>31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0</td>
</tr>
</tbody>
</table>

For PPIs and SGIs, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_enable_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Accessing the GICR_ISENABLER0**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISENABLER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ISENABLER<n> with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by GICD_ISENABLER<n>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISENABLER0 can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
</table>
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GICR_ISENABLER<n>E characteristics are:

### Purpose

Enables forwarding of the corresponding PPI to the CPU interfaces.

### Configuration

Some or all RW fields of this register have defined reset values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_ISENABLER<n>E are RES0.

A copy of this register is provided for each Redistributor.

### Attributes

GICR_ISENABLER<n>E is a 32-bit register.

### Field descriptions

The GICR_ISENABLER<n>E bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Set_enable_bit<x>, bit [x], for x = 0 to 31

For the extended PPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_enable_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1.</td>
</tr>
</tbody>
</table>

This field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISENABLER<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ISENABLER<n>E is (0x100 + (4*n)).
- The bit number of the required group modifier bit in this register is (m-1024) MOD 32.

### Accessing the GICR_ISENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISENABLER<n>E, the corresponding bit is RES0.
When `GICD_CTLR.DS==0`, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICR_ISENABLER<n>E** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0100+4n</td>
<td>GICR_ISENABLER&lt;n&gt;E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RW**.
- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICR_ISPENDR0, Interrupt Set-Pending Register 0

The GICR_ISPENDR0 characteristics are:

**Purpose**

Adds the pending state to the corresponding SGI or PPI.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ISPENDR0 is a 32-bit register.

**Field descriptions**

The GICR_ISPENDR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

*Set_pending_bit<\(x\)*, bit \([x]\), for \(x = 0\) to \(31\)

For PPIs and SGIs, adds the pending state to interrupt number \(x\). Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th><em>Set_pending_bit&lt;(x)&gt;</em></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect.</td>
</tr>
</tbody>
</table>
| 0b1          | If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases:  
  - If the interrupt is already pending because of a write to GICR_ISPENDR0.  
  - If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted. |

This field resets to an architecturally UNKNOWN value.

**Accessing the GICR_ISPENDR0**

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISPENDR0, the corresponding bit is RAZ/WI and equivalent functionality is provided by GICD_ISPENDR<\(n\)> with \(n=0\).

This register only applies to SGIs (bits \([15:0]\)) and PPIs (bits \([31:16]\)). For SPIs, this functionality is provided by GICD_ISPENDR<\(n\)>.

When GICD_CTLR.DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.
**GICR_ISPENDR0 can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0200</td>
<td>GICR_ISPENDR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICR_ISPENDR<n>E, Interrupt Set-Pending Registers, n = 1 - 2

The GICR_ISPENDR<n>E characteristics are:

**Purpose**

Adds the pending state to the corresponding PPI.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_ISPENDR<n>E are RES0.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_ISPENDR<n>E is a 32-bit register.

**Field descriptions**

The GICR_ISPENDR<n>E bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**Set_pending_bit<x>, bit [x], for x = 0 to 31**

For the extended PPIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

<table>
<thead>
<tr>
<th>Set_pending_bit&lt;x&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases:</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is already pending because of a write to GICR_ISPENDR&lt;n&gt;E.</td>
</tr>
<tr>
<td></td>
<td>• If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISPENDR<n>E number, n, is given by n = (m-1024) DIV 32.
- The offset of the required GICR_ISPENDR<n>E is (0x200 + (4*n)).
• The bit number of the required group modifier bit in this register is \((m-1024) \mod 32\).

**Accessing the GICR\_ISPENDR\(_n\)_E**

When affinity routing is not enabled for the Security state of an interrupt in GICR\_ISPENDR\(_n\)_E, the corresponding bit is RES0.

When GICD\_CTLR.DS==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

**GICR\_ISPENDR\(_n\)_E can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x00200 + 4n</td>
<td>GICR_ISPENDR(_n)_E</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When GICD\_CTLR.DS == 0b0 accesses to this register are **RW**.
• When IsAccessSecure() accesses to this register are **RW**.
• When !IsAccessSecure() accesses to this register are **RW**.
GICR_MPAMIDR, Report maximum PARTID and PMG Register

The GICR_MPAMIDR characteristics are:

**Purpose**

Reports the maximum support PARTID and PMG values.

**Configuration**

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_MPAMIDR are RES0.

A copy of this register is provided for each Redistributor.

When GICR_TYPER.MPAM==0, this register is RES0.

**Attributes**

GICR_MPAMIDR is a 32-bit register.

**Field descriptions**

The GICR_MPAMIDR bit assignments are:

```
<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:7</td>
<td>PMGmax, PARTIDmax</td>
</tr>
<tr>
<td>6:0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>
```

**Bits [31:24]**

Reserved, RES0.

**PMGmax, bits [23:16]**

Maximum PMG value supported.

**PARTIDmax, bits [15:0]**

Maximum PARTID value supported.

**Accessing the GICR_MPAMIDR**

**GICR_MPAMIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0018</td>
<td>GICR_MPAMIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The GICR_NSACR characteristics are:

**Purpose**

Enables Secure software to permit Non-secure software to create SGIs targeting the PE connected to this Redistributor by writing to ICC_SGI1R_EL1, ICC_ASGI1R_EL1 or ICC_SGI0R_EL1. See Forwarding an SGI to a target PE for more information.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

For a description on when a write to ICC_SGI0R_EL1, ICC_SGI1R_EL1 or ICC_ASGI1R_EL1 is permitted to generate an interrupt see Use of control registers for SGI forwarding.

**Attributes**

GICR_NSACR is a 32-bit register.

**Field descriptions**

The GICR_NSACR bit assignments are:

<table>
<thead>
<tr>
<th>NS_access&lt;x&gt;, bits [2x+1:2x], for x = 0 to 15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
</tr>
<tr>
<td>0b01</td>
</tr>
<tr>
<td>0b10</td>
</tr>
<tr>
<td>0b11</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Accessing the GICR_NSACR**

When GICD_CTLR.DS == 1, this register is RAZ/WI.
When `GICD_CTLR.DS == 0`, this register is Secure, and is RAZ/WI to Non-secure accesses.

This register is used when affinity routing is enabled. When affinity routing is not enabled for the Security state of the interrupt, `GICD_NSACR<n>` with n=0 provides equivalent functionality.

This register does not support PPIs.

**GICR_NSACR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>SGI_base</td>
<td>0x0E00</td>
<td>GICR_NSACR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `GICD_CTLR.DS == 0b0` accesses to this register are **RW**.
- When `IsAccessSecure()` accesses to this register are **RW**.
- When `!IsAccessSecure()` accesses to this register are **RW**.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICR_PARTIDR, Set PARTID and PMG Register

The GICR_PARTIDR characteristics are:

**Purpose**

Sets the PARTID and PMG values used for memory accesses by the Redistributor.

**Configuration**

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GICR_PARTIDR are RES0. A copy of this register is provided for each Redistributor.

When GICR_TYPER.MPAM==0, this register is RES0.

**Attributes**

GICR_PARTIDR is a 32-bit register.

**Field descriptions**

The GICR_PARTIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>PMG</td>
</tr>
<tr>
<td>29</td>
<td>PARTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**PMG, bits [23:16]**

PMG value used when Redistributor accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PMG values in the range 0 to PMG_MAX are stateful or RES0.

**PARTID, bits [15:0]**

PARTID value used when Redistributor accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PARTID values in the range 0 to PARTID_MAX are stateful or RES0.

**Accessing the GICR_PARTIDR**

GICR_PARTIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC</td>
<td>RD_base</td>
<td>0x001C</td>
<td>GICR_PARTIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GiCD_CTRLS.DS == 0b0 accesses to this register are RW.
• When IsAccessSecure() accesses to this register are **RW**.
• When !IsAccessSecure() accesses to this register are **RW**.
GICR_PENDBASER, Redistributor LPI Pending Table Base Address Register

The GICR_PENDBASER characteristics are:

**Purpose**

Specifies the base address of the LPI Pending table, and the Shareability and Cacheability of accesses to the LPI Pending table.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_PENDBASER is a 64-bit register.

**Field descriptions**

The GICR_PENDBASER bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>
| 62  | PTZ, bit 62

Pending Table Zero. Indicates to the Redistributor whether the LPI Pending table is zero when GICR_CTLR.EnableLPIS == 1.

This field is WO, and reads as 0.

<table>
<thead>
<tr>
<th>PTZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The LPI Pending table is not zero, and contains live data.</td>
</tr>
<tr>
<td>1</td>
<td>The LPI Pending table is zero. Software must ensure the LPI</td>
</tr>
<tr>
<td></td>
<td>Pending table is zero before this value is written.</td>
</tr>
</tbody>
</table>

**Bits [61:59]**

Reserved, RES0.

**OuterCache, bits [58:56]**

Indicates the Outer Cacheability attributes of accesses to the LPI Pending table. The possible values of this field are:
OuterCache

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Outer Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Outer Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Outer Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Outer Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:16]

Bits [51:16] of the physical address containing the LPI Pending table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

This field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Pending table. The possible values of this field are:

<table>
<thead>
<tr>
<th>Shareability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b001</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b110</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b111</td>
<td>Reserved. Treated as 0b00.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Pending table. The possible values of this field are:

<table>
<thead>
<tr>
<th>InnerCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Inner Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Inner Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Inner Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Inner Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>
This field resets to an architecturally **UNKNOWN** value.

**Bits [6:0]**

Reserved, **RES0**.

**Accessing the GICR_PENDBASER**

Having the GICR_PENDBASER OuterCache, Shareability or InnerCache fields programmed to different values on different Redistributors with **GICR_CTLR**.EnableLPIs == 1 in the system is **UNPREDICTABLE**.

Changing GICR_PENDBASER with **GICR_CTLR**.EnableLPIs == 1 is **UNPREDICTABLE**.

**GICR_PENDBASER can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0078</td>
<td>GICR_PENDBASER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICR_PROPBASE, Redistributor Properties Base Address Register

The GICR_PROPBASE characteristics are:

**Purpose**

Specifies the base address of the LPI Configuration table, and the Shareability and Cacheability of accesses to the LPI Configuration table.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

A copy of this register is provided for each Redistributor.

An implementation might make this register RO, for example to correspond to an LPI Configuration table in read-only memory.

**Attributes**

GICR_PROPBASE is a 64-bit register.

**Field descriptions**

The GICR_PROPBASE bit assignments are:

<table>
<thead>
<tr>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63:59</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**OuterCache, bits [58:56]**

Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table. The possible values of this field are:

<table>
<thead>
<tr>
<th>OuterCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Outer Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Outer Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Outer Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Outer Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

It is **IMPLEMENTATION DEFINED** whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.
This field resets to an architecturally **UNKNOWN** value.

**Bits [55:52]**

Reserved, RES0.

**Physical Address, bits [51:12]**

Bits [51:12] of the physical address containing the LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

This field resets to an architecturally **UNKNOWN** value.

**Shareability, bits [11:10]**

Indicates the Shareability attributes of accesses to the LPI Configuration table. The possible values of this field are:

<table>
<thead>
<tr>
<th>Shareability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Reserved. Treated as 0b00.</td>
</tr>
</tbody>
</table>

It is **IMPLEMENTATION DEFINED** whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally **UNKNOWN** value.

**InnerCache, bits [9:7]**

Indicates the Inner Cacheability attributes of accesses to the LPI Configuration table. The possible values of this field are:

<table>
<thead>
<tr>
<th>InnerCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Inner Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Inner Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Inner Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Inner Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate,</td>
</tr>
<tr>
<td></td>
<td>Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate,</td>
</tr>
<tr>
<td></td>
<td>Write-back.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

**Bits [6:5]**

Reserved, RES0.

**IDbits, bits [4:0]**

The number of bits of LPI INTID supported, minus one, by the LPI Configuration table starting at Physical Address.

If the value of this field is larger than the value of `GICD_TYPER.IDbits`, the `GICD_TYPER.IDbits` value applies.

If the value of this field is less than 0b1101, indicating that the largest INTID is less than 8192 (the smallest LPI interrupt ID), the GIC will behave as if all physical LPIs are out of range.

This field resets to an architecturally **UNKNOWN** value.
Accessing the GICR_PROPBASER

It is IMPLEMENTATION DEFINED whether GICR_PROPBASER can be set to different values on different Redistributors. GICR_TYPER.CommonLPIAff identifies the Redistributors that must have GICR_PROPBASER set to the same values whenever GICR_CTLR.EnableLPis == 1.

Setting different values in different copies of GICR_PROPBASER on Redistributors that are required to use a common LPI Configuration table when GICR_CTLR.EnableLPis == 1 leads to UNPREDICTABLE behavior.

Other restrictions apply when a Redistributor caches information from GICR_PROPBASER. See LPI Configuration tables for more information.

**GICR_PROPBASER can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0070</td>
<td>GICR_PROPBASER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GICR_SETLPIR, Set LPI Pending Register

The GICR_SETLPIR characteristics are:

**Purpose**

Generates an LPI by setting the pending state of the specified LPI.

**Configuration**

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_SETLPIR is a 64-bit register.

**Field descriptions**

The GICR_SETLPIR bit assignments are:

| Bit 63 | Bit 62 | Bit 61 | Bit 60 | Bit 59 | Bit 58 | Bit 57 | Bit 56 | Bit 55 | Bit 54 | Bit 53 | Bit 52 | Bit 51 | Bit 50 | Bit 49 | Bit 48 | Bit 47 | Bit 46 | Bit 45 | Bit 44 | Bit 43 | Bit 42 | Bit 41 | Bit 40 | Bit 39 | Bit 38 | Bit 37 | Bit 36 | Bit 35 | Bit 34 | Bit 33 | Bit 32 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        |        |        |        | RES0   | pINTID |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |        |

**Bits [63:32]**

Reserved, RES0.

**pINTID, bits [31:0]**

The INTID of the physical LPI to be generated.

**Note**

The size of this field is IMPLEMENTATION DEFINED, and is specified by the GICD_TYPER.IDbits field. Unimplemented bits are RES0.

**Accessing the GICR_SETLPIR**

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if either:

- The pINTID field corresponds to an LPI that is already pending.
- The pINTID field corresponds to an unimplemented LPI.
- GICR_CTLR_EnableLPIs == 0.

GICR_SETLPIR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GICR_SETLPIR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Page 3778
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are WO.
- When IsAccessSecure() accesses to this register are WO.
- When !IsAccessSecure() accesses to this register are WO.
The GICR_STATUSR characteristics are:

**Purpose**

Provides software with a mechanism to detect:

- **Accesses to reserved locations.**
- **Writes to read-only locations.**
- **Reads of write-only locations.**

**Configuration**

A copy of this register is provided for each Redistributor.

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

**Attributes**

GICR_STATUSR is a 32-bit register.

**Field descriptions**

The GICR_STATUSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>31</td>
<td>WROD, bit 3</td>
</tr>
<tr>
<td>2</td>
<td>RWOD, bit 2</td>
</tr>
<tr>
<td>1</td>
<td>WRD, bit 1</td>
</tr>
</tbody>
</table>

**WROD, bit [3]**

Write to an RO location.

<table>
<thead>
<tr>
<th>WROD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write to an RO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**RWOD, bit [2]**

Read of a WO location.

<table>
<thead>
<tr>
<th>RWOD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a WO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**WRD, bit [1]**

Write to a reserved location.
When a violation is detected, software must write 1 to this register to reset it.

**RRD, bit [0]**

Read of a reserved location.

<table>
<thead>
<tr>
<th>RRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**Accessing the GICR_STATUSR**

This is an optional register. If the register is not implemented, the location is RAZ/WI.

**GICR_STATUSR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0010</td>
<td>GICR_STATUSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0010</td>
<td>GICR_STATUSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICR_SYNCR, Redistributor Synchronize Register

The GICR_SYNCR characteristics are:

Purpose

Indicates completion of physical Redistributor operations.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_SYNCR is a 32-bit register.

Field descriptions

The GICR_SYNCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Busy</td>
<td>Indicates completion of any Redistributor operations as follows:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Busy</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No operations are in progress.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write is in progress to one or more of the following registers:</td>
</tr>
<tr>
<td></td>
<td>• <strong>GICR_CLRLPIR</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>GICR_INVLPPIR</strong>.</td>
</tr>
<tr>
<td></td>
<td>• <strong>GICR_INVALLR</strong>.</td>
</tr>
</tbody>
</table>

This field also indicates completion of any operations initiated by writes to **GICR_PENDBASER** or **GICR_PROPBASER**.

Accessing the GICR_SYNCR

Optionally, when this register is accessed, an implementation might wait until all operations are complete before returning a value, in which case GICR_SYNCR.Busy is always 0.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

GICR_SYNCR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redistributor</td>
<td>RD_base</td>
<td>0x00C0</td>
<td>GICR_SYNCR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
- When GICD_CTRL.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.
The GICR_TYPER characteristics are:

**Purpose**

Provides information about the configuration of this Redistributor.

**Configuration**

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_TYPER is a 64-bit register.

**Field descriptions**

The GICR_TYPER bit assignments are:

<table>
<thead>
<tr>
<th>Affinity_Value</th>
<th>PPInum</th>
<th>CommonLPIAff</th>
<th>Processor_Number</th>
<th>MPAM</th>
<th>DPGS</th>
<th>LastDirectLPI</th>
<th>Dirty</th>
<th>VLPIS</th>
<th>PLPIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>63</td>
<td>58</td>
<td>57</td>
<td>56</td>
<td>55</td>
<td>54</td>
<td>53</td>
<td>52</td>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>32</td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
</tr>
<tr>
<td>30</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>29</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>28</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>25</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>24</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>23</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>21</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>19</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
</tbody>
</table>

**Affinity_Value, bits [63:32]**

The identity of the PE associated with this Redistributor.

- Bits [63:56] provide Aff3, the Affinity level 3 value for the Redistributor.
- Bits [55:48] provide Aff2, the Affinity level 2 value for the Redistributor.
- Bits [47:40] provide Aff1, the Affinity level 1 value for the Redistributor.
- Bits [39:32] provide Aff0, the Affinity level 0 value for the Redistributor.

**PPInum, bits [31:27]**

*When GIC, >=3.1 is implemented:*

The value derived from this field specifies the maximum PPI INTID that a GIC implementation might support. An implementation might not implement all PPIs up to this maximum.

<table>
<thead>
<tr>
<th>PPInum</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Maximum PPI INTID is 31.</td>
</tr>
<tr>
<td>0b00001</td>
<td>Maximum PPI INTID is 1087.</td>
</tr>
<tr>
<td>0b00010</td>
<td>Maximum PPI INTID is 1119.</td>
</tr>
</tbody>
</table>

All other values are reserved.

*Otherwise:*

Reserved, RES0.
Bit [26]

Reserved, RES0.

CommonLPIAff, bits [25:24]

The affinity level at which Redistributors share a LPI Configuration table.

<table>
<thead>
<tr>
<th>CommonLPIAff</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>All Redistributors must share an LPI Configuration table.</td>
</tr>
<tr>
<td>0b01</td>
<td>All Redistributors with the same Aff3 value must share an LPI Configuration table.</td>
</tr>
<tr>
<td>0b10</td>
<td>All Redistributors with the same Aff3.Aff2 value must share an LPI Configuration table.</td>
</tr>
<tr>
<td>0b11</td>
<td>All Redistributors with the same Aff3.Aff2.Aff1 value must share an LPI Configuration table.</td>
</tr>
</tbody>
</table>

Processor_Number, bits [23:8]

A unique identifier for the PE. When GITS_TYPER.PTA == 0, an ITS uses this field to identify the interrupt target.

When affinity routing is disabled for a Security state, this field indicates which GICD_ITARGETSR<n> corresponds to this Redistributor.

Bit [7]

Reserved, RES0.

MPAM, bit [6]

When GIC, >=3.1 is implemented:

MPAM

<table>
<thead>
<tr>
<th>MPAM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAM not supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM supported.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

DPGS, bit [5]

Sets support for GICR_CTLR.DPG* bits.

<table>
<thead>
<tr>
<th>DPGS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICR_CTLR.DPG* bits are not supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICR_CTLR.DPG* bits are supported.</td>
</tr>
</tbody>
</table>

Last, bit [4]

Indicates whether this Redistributor is the highest-numbered Redistributor in a series of contiguous Redistributor pages.

<table>
<thead>
<tr>
<th>Last</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This Redistributor is not the highest-numbered Redistributor in a series of contiguous Redistributor pages.</td>
</tr>
<tr>
<td>0b1</td>
<td>This Redistributor is the highest-numbered Redistributor in a series of contiguous Redistributor pages.</td>
</tr>
</tbody>
</table>
**DirectLPI, bit [3]**

Indicates whether this Redistributor supports direct injection of LPIs.

<table>
<thead>
<tr>
<th>DirectLPI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This Redistributor does not support direct injection of LPIs. The GICR_SETLPIR, GICR_CLRPLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR registers are either not implemented, or have an IMPLEMENTATION_DEFINED purpose.</td>
</tr>
<tr>
<td>0b1</td>
<td>This Redistributor supports direct injection of LPIs. The GICR_SETLPIR, GICR_CLRPLPIR, GICR_INVLPIR, GICR_INVALLR, and GICR_SYNCR registers are implemented.</td>
</tr>
</tbody>
</table>

**Dirty, bit [2]**

Controls the functionality of GICR_VPENDBASER.Dirty.

<table>
<thead>
<tr>
<th>Dirty</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICR_VPENDBASER.Dirty is UNKNOWN when GICR_VPENDBASER.Valid == 1.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICR_VPENDBASER.Dirty indicates when the Virtual Pending Table has been parsed when GICR_VPENDBASER.Valid is written from 0 to 1.</td>
</tr>
</tbody>
</table>

When GICR_TYPER.VLPIS == 0, this field is RES0.

**VLPIS, bit [1]**

Indicates whether the GIC implementation supports virtual LPIs and the direct injection of virtual LPIs.

<table>
<thead>
<tr>
<th>VLPIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support virtual LPIs or the direct injection of virtual LPIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports virtual LPIs and the direct injection of virtual LPIs.</td>
</tr>
</tbody>
</table>

**Note**

In GICv3 implementations this field is RES0.

**PLPIS, bit [0]**

Indicates whether the GIC implementation supports physical LPIs.

<table>
<thead>
<tr>
<th>PLPIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support physical LPIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports physical LPIs.</td>
</tr>
</tbody>
</table>

**Accessing the GICR_TYPER**

**GICR_TYPER can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0008</td>
<td>GICR_TYPER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GICR_VPENDBASER, Virtual Redistributor LPI Pending Table Base Address Register

The GICR_VPENDBASER characteristics are:

**Purpose**

Specifies the base address of the memory that holds the virtual LPI Pending table for the currently scheduled virtual machine.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is provided only in GICv4 implementations.

**Attributes**

GICR_VPENDBASER is a 64-bit register.

**Field descriptions**

The GICR_VPENDBASER bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Valid</th>
<th>IDAI</th>
<th>Pending Last</th>
<th>Dirty</th>
<th>RES0</th>
<th>OuterCache</th>
<th>Physical Address</th>
<th>RES0</th>
<th>Physical Address</th>
<th>RES0</th>
<th>Shareability</th>
<th>InnerCache</th>
<th>RES0</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Valid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td></td>
<td>IDAI</td>
<td></td>
<td>Dirty</td>
<td>RES0</td>
<td>OuterCache</td>
<td>Physical Address</td>
<td>RES0</td>
<td>Shareability</td>
<td>InnerCache</td>
<td>RES0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Pending Last</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
</tr>
</tbody>
</table>

This bit controls whether the virtual LPI Pending table is valid:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The virtual LPI Pending table is not valid. No vPE is scheduled.</td>
</tr>
<tr>
<td>0b1</td>
<td>The virtual LPI Pending table is valid. A vPE is scheduled.</td>
</tr>
</tbody>
</table>

Setting GICR_VPENDBASER.Valid == 1 when the associated CPU interface does not implement GICv4 is UNPREDICTABLE.

**Note**

Software can determine whether a PE supports GICv3 or GICv4 by reading ID_AA64PFR0_EL1.

Writing a new value to any bit of GICR_VPENDBASER, other than GICR_VPENDBASER.Valid, when GICR_VPENDBASER.Valid==1 is UNPREDICTABLE.

This field resets to 0.

**IDAI, bit [62]**

Implementation Defined Area Invalid. Indicates whether the IMPLEMENTATION DEFINED area in the virtual LPI Pending table is valid:
For more information, see LPI Pending tables and Virtual LPI Configuration tables and virtual LPI Pending tables.

This field resets to an architecturally UNKNOWN value.

**PendingLast, bit [61]**

Indicates whether there are pending and enabled interrupts for the last scheduled vPE.

This value is set by the implementation when GICR_VPENDBASER.Valid has been written from 1 to 0 and is otherwise UNKNOWN.

<table>
<thead>
<tr>
<th>PendingLast</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There are no pending and enabled interrupts for the last scheduled vPE.</td>
</tr>
<tr>
<td>0b1</td>
<td>There is at least one pending interrupt for the last scheduled vPE. It is IMPLEMENTATION DEFINED whether this bit is set when the only pending interrupts for the last scheduled vPE are not enabled. Arm deprecates setting PendingLast to 1 when the only pending interrupts for the last scheduled virtual machine are not enabled.</td>
</tr>
</tbody>
</table>

When the GICR_VPENDBASER.Valid bit is written from 0 to 1, this bit is RES1.

This field resets to 0.

**Dirty, bit [60]**

*When GICR_VPENDBASER.Valid == 0b0:*

This is field is read-only. Indicates whether a de-scheduling operation is in process.

<table>
<thead>
<tr>
<th>Dirty</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No de-scheduling operation in process.</td>
</tr>
<tr>
<td>0b1</td>
<td>De-scheduling operation in process.</td>
</tr>
</tbody>
</table>

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty==1.

This field resets to 0.

*When GICR_VPENDBASER.Valid == 0b1 and GICR_TYPER.Dirty == 0b1:*

This field is read-only. Reports whether the Virtual Pending table has been parsed.

<table>
<thead>
<tr>
<th>Dirty</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Parsing of the Virtual Pending Table has completed.</td>
</tr>
<tr>
<td>0b1</td>
<td>Parsing of the Virtual Pending Table has not completed.</td>
</tr>
</tbody>
</table>

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty==1.

This field resets to 0.

**Otherwise:**

This field is read-only. This fields is UNKNOWN.

This field resets to 0.
Bit [59]
Reserved, RES0.

OuterCache, bits [58:56]
Indicates the Outer Cacheability attributes of accesses to virtual LPI Pending tables of vPEs targeting this Redistributor. The possible values of this field are:

<table>
<thead>
<tr>
<th>OuterCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Outer Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Outer Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Outer Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Outer Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the OuterCacheability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

This field resets to an architecturally UNKNOWN value.

Bits [55:52]
Reserved, RES0.

Physical_Address, bits [51:16]

Bits [51:16] of the physical address containing the virtual LPI Pending table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

This field resets to an architecturally UNKNOWN value.

Bits [15:12]
Reserved, RES0.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the virtual LPI Pending table. The possible values of this field are:

<table>
<thead>
<tr>
<th>Shareability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Reserved. Treated as 0b00.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.
If the Shareability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is *UNPREDICTABLE*.

This field resets to an architecturally *UNKNOWN* value.

### InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the virtual LPI Pending table. The possible values of this field are:

<table>
<thead>
<tr>
<th>InnerCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Inner Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Inner Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Inner Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Inner Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the InnerCacheability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is *UNPREDICTABLE*.

This field resets to an architecturally *UNKNOWN* value.

### Bits [6:0]

Reserved, RES0.

### Accessing the GICR_VPENDBASER

The effect of a write to this register is not guaranteed to be visible throughout the affinity hierarchy, as indicated by GICR_CTLR.RWP == 0.

**GICR_VPENDBASER** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redistributor</td>
<td>VLPI_base</td>
<td>0x0078</td>
<td>GICR_VPENDBASER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICR_VPROPBASER, Virtual Redistributor Properties
Base Address Register

The GICR_VPROPBASER characteristics are:

**Purpose**

Specifies the base address of the memory that holds the virtual LPI Configuration table for the currently scheduled virtual machine.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is provided in GICv4 implementations only.

**Attributes**

GICR_VPROPBASER is a 64-bit register.

**Field descriptions**

The GICR_VPROPBASER bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, <strong>RES0</strong></td>
</tr>
<tr>
<td>58-56</td>
<td>Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table. The possible values of this field are:</td>
</tr>
<tr>
<td>31-20</td>
<td>Reserved, <strong>RES0</strong></td>
</tr>
<tr>
<td>19-12</td>
<td>Shareability, <strong>InnerCache</strong>, <strong>RES0</strong>, <strong>IDbits</strong></td>
</tr>
<tr>
<td>11-0</td>
<td><strong>Physical_Address</strong></td>
</tr>
</tbody>
</table>

**OuterCache, bits [58:56]**

Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table. The possible values of this field are:

<table>
<thead>
<tr>
<th>OuterCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Outer Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Outer Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Outer Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Outer Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

It is **IMPLEMENTATION DEFINED** whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally **UNKNOWN** value.
Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the virtual LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

This field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Configuration table. The possible values of this field are:

<table>
<thead>
<tr>
<th>Shareability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Reserved. Treated as 0b00.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Configuration table. The possible values of this field are:

<table>
<thead>
<tr>
<th>InnerCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Inner Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Inner Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Inner Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Inner Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IDbits, bits [4:0]

The number of bits of virtual LPI INTID supported, minus one.

If the value of this field is less than 0b1101, indicating that the largest INTID is less than 8192 (the smallest LPI interrupt ID), the GIC will behave as if all virtual LPIs are out of range.

This field resets to an architecturally UNKNOWN value.
Accessing the **GICR_VPROPBASER**

**GICR_VPROPBASER** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC</td>
<td>VLPI_base</td>
<td>0x0070</td>
<td>GICR_VPROPBASER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The GICR_WAKER characteristics are:

**Purpose**

Permits software to control the behavior of the WakeRequest power management signal corresponding to the Redistributor. Power management operations follow the rules in Power management.

**Configuration**

Some or all RW fields of this register have defined reset values.

A copy of this register is provided for each Redistributor.

**Attributes**

GICR_WAKER is a 32-bit register.

**Field descriptions**

The GICR_WAKER bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
<tr>
<td>30:29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>ChildrenAsleep</td>
</tr>
<tr>
<td>27</td>
<td>ProcessorSleep</td>
</tr>
<tr>
<td>26:0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bit [31]**

IMPLEMENTATION DEFINED.

**Bits [30:3]**

Reserved, RES0.

**ChildrenAsleep, bit [2]**

Read-only. Indicates whether the connected PE is quiescent:

<table>
<thead>
<tr>
<th>ChildrenAsleep</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An interface to the connected PE might be active.</td>
</tr>
<tr>
<td>0b1</td>
<td>All interfaces to the connected PE are quiescent.</td>
</tr>
</tbody>
</table>

This field resets to 1.

**ProcessorSleep, bit [1]**

Indicates whether the Redistributor can assert the WakeRequest signal:
### ProcessorSleep

<table>
<thead>
<tr>
<th>ProcessorSleep</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This PE is not in, and is not entering, a low power state.</td>
</tr>
</tbody>
</table>
| 0b1            | The PE is either in, or is in the process of entering, a low power state. All interrupts that arrive at the Redistributor:  
  - Assert a WakeRequest signal.  
  - Are held in the pending state at the Redistributor, and are not communicated to the CPU interface. |

**Note**

When ProcessorSleep == 1, the Redistributor must ensure that any interrupts that are pending on the CPU interface are released.

For an implementation that is using the GIC Stream Protocol Interface:

- A Quiesce command can put the interface between the Redistributor and the CPU interface in a quiescent state.
- A Release command can release any interrupts that are pending on the CPU interface.

**Note**

Before powering down a PE, software must set this bit to 1 and wait until ChildrenAsleep == 1. After powering up a PE, or following a failed powerdown, software must set this bit to 0 and wait until ChildrenAsleep == 0.

Changing ProcessorSleep from 1 to 0 when ChildrenAsleep is not 1 results in **UNPREDICTABLE** behavior.

Changing ProcessorSleep from 0 to 1 when the Enable for each interrupt group in the associated CPU interface is not 0 results in **UNPREDICTABLE** behavior.

This field resets to 1.

**IMPLEMENTATION DEFINED, bit [0]**

**IMPLEMENTATION DEFINED.**

### Accessing the GICR_WAKER

When GICD_CTLR.DS==1, this register is always accessible.

When GICD_CTLR.DS==0, this is a Secure register. This register is RAZ/WI to Non-secure accesses.

To ensure a Redistributor is quiescent, software must write to GICR_WAKER with ProcessorSleep == 1, then poll the register until ChildrenAsleep == 1.

Resetting the connected PE when GICR_WAKER.ProcessorSleep==0 or GICR_WAKER.ChildrenAsleep==0, can lead to **UNPREDICTABLE** behaviour in the IRI.

Resetting the IRI when GICR_WAKER.ProcessorSleep==0 or GICR_WAKER.ChildrenAsleep==0 can lead to **UNPREDICTABLE** behaviour in the connected PE.

**GICR_WAKER can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Redistributor</td>
<td>RD_base</td>
<td>0x0014</td>
<td>GICR_WAKER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When GICD_CTLR.DS == 0b0 accesses to this register are RW.
• When IsAccessSecure() accesses to this register are RW.
GICV_ABPR, Virtual Machine Aliased Binary Point Register

The GICV_ABPR characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

This register corresponds to **GICC_ABPR** in the physical CPU interface.

---

**Note**

**GICH_LR<n>**.Group determines whether a virtual interrupt is Group 0 or Group 1.

---

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is available when the GIC implementation supports interrupt virtualization.

---

**Attributes**

GICV_ABPR is a 32-bit register.

---

**Field descriptions**

The GICV_ABPR bit assignments are:

```
    31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

**Bits [31:3]**

Reserved, RES0.

**Binary_Point, bits [2:0]**

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For information about how this field determines the interrupt priority bits assigned to the group priority field, see Priority grouping.

This field resets to 0.

The Binary_Point field of this register is aliased to **GICH_VMCR.VBPR1**.

---

**Accessing the GICV_ABPR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, **ICC_BPR1** provides equivalent functionality.
- For AArch64 implementations, **ICC_BPR1_EL1** provides equivalent functionality.
The value contained in this register is one greater than the actual applied binary point value, as described in 'Priority grouping' in the GICv3 Architecture Specification.

This register is used for Group 1 interrupts when GICV_CTLR.CBPR == 0. GICV_BPR provides equivalent functionality for Group 0 interrupts, and for Group 1 interrupts when GICV_CTLR.CBPR == 1.

**GICV_ABPR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x001C</td>
<td>GICV_ABPR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GICV_AEOIR, Virtual Machine Aliased End Of Interrupt Register

The GICV_AEOIR characteristics are:

**Purpose**

A write to this register performs a priority drop for the specified Group 1 virtual interrupt and, if GICV_CTLR.EOImode == 0, also deactivates the interrupt.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_AEOIR is a 32-bit register.

**Field descriptions**

The GICV_AEOIR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:25]**

Reserved, RES0.

**INTID, bits [24:0]**

The INTID of the signaled interrupt.

---

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

---

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

A successful EOI request means that:

- The highest priority bit in GICH_APR<\text{n}> is cleared, causing the running priority to drop.
- If the appropriate GICV_CTLR EOImode bit == 0, the interrupt is deactivated in the corresponding List register. If the INTID corresponds to a hardware interrupt, the interrupt is also deactivated in the Distributor.

---

**Note**

Only Group 1 interrupts can target the hypervisor, and therefore only Group 1 interrupts are deactivated in the Distributor.
A write to this register is \textit{UNPREDICTABLE} if the INTID corresponds to a Group 0 interrupt. In addition, the following GICv2 \textit{UNPREDICTABLE} cases require specific actions:

- If highest active priority is Group 0 and the identified interrupt is in the List Registers and it matches the highest active priority. When EL2 is using System registers and \texttt{ICH_VTR_EL2.SEIS} is 1, an \textit{IMPLEMENTATION DEFINED} SEI might be generated, otherwise GICv3 implementations must ignore such writes.
- If the identified interrupt is in the List Registers, and the HW bit is 1, and the interrupt to be deactivated is an SGI (that is, the value of Physical ID is between 0 and 15). GICv3 implementations must perform the deactivate operation. This means that a GICv3 implementation in legacy operation must ensure only a single SGI is active for a PE.
- If the identified interrupt is in the List Registers, and the HW bit is 1, and the corresponding pINTID field value is between 1020 and 1023, indicating a special purpose INTID. GICv3 implementations must not perform a deactivate operation but must still change the state of the List register as appropriate. When EL2 is using System registers and \texttt{ICH_VTR_EL2.SEIS} is 1, an implementation might generate a system error.

### Accessing the GICV_AEOIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, \texttt{ICC_EOIR1} provides equivalent functionality.
- For AArch64 implementations, \texttt{ICC_EOIR1_EL1} provides equivalent functionality.

This register is used for Group 1 interrupts only. \texttt{GICV_EOIR} provides equivalent functionality for Group 0 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

### GICV_AEOIR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU</td>
<td>0x0024</td>
<td>GICV_AEOIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{GICD_CTLR.DS == 0b0} accesses to this register are \texttt{WO}.
- When \texttt{IsAccessSecure()} accesses to this register are \texttt{WO}.
- When \texttt{!IsAccessSecure()} accesses to this register are \texttt{WO}.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
GICV_AHPPIR, Virtual Machine Aliased Highest Priority Pending Interrupt Register

The GICV_AHPPIR characteristics are:

**Purpose**

Provides the INTID of the highest priority pending Group 1 virtual interrupt in the List registers.

This register corresponds to the physical CPU interface register `GICC_AHPPIR`.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_AHPPIR is a 32-bit register.

**Field descriptions**

The GICV_AHPPIR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>25-0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:25]**

Reserved, RES0.

**INTID, bits [24:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

A read of this register returns the spurious INTID 1023 if any of the following are true:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE.
- The highest priority pending interrupt is in Group 0.

**Accessing the GICV_AHPPIR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, `ICC_HPPIR1` provides equivalent functionality.
- For AArch64 implementations, `ICC_HPPIR1_EL1` provides equivalent functionality.
This register is used for Group 1 interrupts only. GICV_HPPIR provides equivalent functionality for Group 0 interrupts.

The register does not return the INTID of an interrupt that is active and pending.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

**GICV_AHPPIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU</td>
<td>0x0028</td>
<td>GICV_AHPPIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.
The GICV_AIAR characteristics are:

**Purpose**

Provides the INTID of the signaled Group 1 virtual interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

This register corresponds to the physical CPU interface register **GICC_AIAR**.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_AIAR is a 32-bit register.

**Field descriptions**

The GICV_AIAR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>INTID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:25]**

Reserved, RES0.

**INTID, bits [24:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

The operation of this register is similar to the operation of **GICV_IAR**. When a vPE reads this register, the corresponding **GICH_LR<n>**.Group field is checked to determine whether the interrupt is in Group 0 or Group 1:

- If the interrupt is Group 0, the spurious INTID 1023 is returned and the interrupt is not acknowledged.
- If the interrupt is Group 1, the INTID is returned. The List register entry is updated to active state, and the appropriate bit in **GICH_APR<n>** is set to 1.

A read of this register returns the spurious INTID 1023 if any of the following are true:

- When the virtual CPU interface is enabled and **GICH_HCR.En == 1**:
  - There are no pending interrupts of sufficiently high priority value to be signaled to the PE.
The highest priority pending interrupt is in Group 0.

- Interrupt signaling by the virtual CPU interface is disabled.

**Accessing the GICV_AIAR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_IAR1 provides equivalent functionality.
- For AArch64 implementations, ICC_IAR1_EL1 provides equivalent functionality.

This register is used for Group 1 interrupts only. GICV_IAR provides equivalent functionality for Group 0 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

**GICV_AIAR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU</td>
<td>0x0020</td>
<td>GICV_AIAR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
**GICV_APR<n>, Virtual Machine Active Priorities Registers, n = 0 - 3**

The GICV_APR<n> characteristics are:

**Purpose**

Provides information about interrupt active priorities.

These registers correspond to the physical CPU interface registers GICC_APR<n>.

**Configuration**

When System register access is disabled for EL2, these registers access GICH_APR<n>, and all active priorities for virtual machines are held in GICH_APR<n> regardless of interrupt group.

When System register access is enabled for EL2, these registers access ICH_AP1R<n>_EL2, and all active priorities for virtual machines are held in ICH_AP1R<n>_EL2 regardless of interrupt group.

**Attributes**

GICV_APR<n> is a 32-bit register.

**Field descriptions**

The GICV_APR<n> bit assignments are:

```
```

P<x>, bit [x], for x = 0 to 31

Provides information about active priorities for the virtual machine.

See GICH_APR<n> and ICH_AP1R<n>_EL2 for the correspondence between priorities and bits.

**Accessing the GICV_APR<n>**

If System register access is not enabled for EL2, these registers access GICH_APR<n>. If System register access is enabled for EL2, these registers access ICH_AP1R<n>_EL2. All active priority mapped guests are held in the accessed registers, regardless of interrupt group.

**GICV_APR<n> can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x00D0 + 4n</td>
<td>GICV_APR&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GICV_BPR characteristics are:

**Purpose**

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

This register corresponds to **GICC_BPR** in the physical CPU interface.

---

**Note**

GICH_LR<n> Group determines whether a virtual interrupt is Group 0 or Group 1.

---

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is available when the GIC implementation supports interrupt virtualization.

When **GICV_CTLR**.CBPR == 1, this register determines interrupt preemption for both Group 0 and Group 1 interrupts.

**Attributes**

GICV_BPR is a 32-bit register.

**Field descriptions**

The GICV_BPR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Binary_Point |

**Bits [31:3]**

Reserved, RES0.

**Binary_Point, bits [2:0]**

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For information about how this field determines the interrupt priority bits assigned to the group priority field, see Priority grouping for Group 0 interrupts, or Group 1 interrupts when CBPR==1

This field resets to an architecturally UNKNOWN value.

The Binary_Point field of this register is aliased to **GICH_VMCR**.VBPR0.

**Accessing the GICV_BPR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, **ICC_BPR0** provides equivalent functionality.
- For AArch64 implementations, **ICC_BPR0_EL1** provides equivalent functionality.
GICV_BPR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU</td>
<td>0x0008</td>
<td>GICV_BPR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GICV_CTLR, Virtual Machine Control Register

The GICV_CTLR characteristics are:

**Purpose**

Controls the behavior of virtual interrupts.

This register corresponds to the physical CPU interface register GICC_CTLR.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

This register is available when a GIC implementation supports interrupt virtualization.

**Attributes**

GICV_CTLR is a 32-bit register.

**Field descriptions**

The GICV_CTLR bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
  RES0  EOImode  RES0  CBPR  FIQEn  AckCtl  EnableGrp1  EnableGrp0
```

**Bits [31:10]**

Reserved, RES0.

**EOImode, bit [9]**

Controls the behavior associated with the GICV_EOIR, GICV_AEOIR, and GICV_DIR registers:

<table>
<thead>
<tr>
<th>EOImode</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Writes to GICV_EOIR and GICV_AEOIR perform priority drop and deactivate interrupt operations simultaneously. Behavior on a write to GICV_DIR is unpredictable. When it has completed processing the interrupt, the virtual machine writes to GICV_EOIR or GICV_AEOIR to deactivate the interrupt. The write updates the List registers and causes the virtual CPU interface to signal the interrupt completion to the physical Distributor.</td>
</tr>
<tr>
<td>0b1</td>
<td>Writes to GICV_EOIR and GICV_AEOIR perform priority drop operation only. Writes to GICV_DIR perform deactivate interrupt operation only. When it has completed processing the interrupt, the virtual machine writes to GICV_DIR to deactivate the interrupt. The write updates the List registers and causes the virtual CPU interface to signal the interrupt completion to the Distributor.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Bits [8:5]**

Reserved, RES0.
CBPR, bit [4]

Controls whether GICV_BPR affects both Group 0 and Group 1 interrupts:

<table>
<thead>
<tr>
<th>CBPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>GICV_BPR affects Group 0 virtual interrupts only. GICV_ABPR affects Group 1 virtual interrupts only.</td>
</tr>
<tr>
<td>0b1</td>
<td>GICV_BPR affects both Group 0 and Group 1 virtual interrupts.</td>
</tr>
</tbody>
</table>

See Priority grouping for more information.

This field resets to an architecturally **UNKNOWN** value.

FIQEn, bit [3]

FIQ Enable. Controls whether Group 0 virtual interrupts are presented as virtual FIQs:

<table>
<thead>
<tr>
<th>FIQEn</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Group 0 virtual interrupts are presented as virtual IRQs.</td>
</tr>
<tr>
<td>0b1</td>
<td>Group 0 virtual interrupts are presented as virtual FIQs.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

AckCtl, bit [2]

Arm deprecates use of this bit. Arm strongly recommends that software is written to operate with this bit always cleared to 0.

Acknowledge control. When the highest priority interrupt is Group 1, determines whether GICV_IAR causes the CPU interface to acknowledge the interrupt or returns the spurious identifier 1022, and whether GICV_HPPIR returns the interrupt ID or the special identifier 1022.

<table>
<thead>
<tr>
<th>AckCtl</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an interrupt ID of 1022.</td>
</tr>
<tr>
<td>0b1</td>
<td>If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the interrupt ID of the corresponding interrupt.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

EnableGrp1, bit [1]

Enables the signaling of Group 1 virtual interrupts by the virtual CPU interface to the virtual machine:

<table>
<thead>
<tr>
<th>EnableGrp1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Signaling of Group 1 interrupts is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Signaling of Group 1 interrupts is enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

EnableGrp0, bit [0]

Enables the signaling of Group 0 virtual interrupts by the virtual CPU interface to the virtual machine:

<table>
<thead>
<tr>
<th>EnableGrp0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Signaling of Group 0 interrupts is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Signaling of Group 0 interrupts is enabled.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.

Accessing the GICV_CTLR

This register is used only when System register access is not enabled. When System register access is enabled:
• For AArch32 implementations, **ICC_CTLR** provides equivalent functionality.
• For AArch64 implementations, **ICC_CTLR_EL1** provides equivalent functionality.

**GICV_CTLR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU</td>
<td>0x0000</td>
<td>GICV_CTLR</td>
</tr>
<tr>
<td>interface</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
• When IsAccessSecure() accesses to this register are **RW**.
• When !IsAccessSecure() accesses to this register are **RW**.
GICV_DIR, Virtual Machine Deactivate Interrupt Register

The GICV_DIR characteristics are:

**Purpose**

Deactivates a specified virtual interrupt in the GICH_LR<n> List registers.
This register corresponds to the physical CPU interface register GICC_DIR.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_DIR is a 32-bit register.

**Field descriptions**

The GICV_DIR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTID |

**Bits [31:25]**

Reserved, RES0.

**INTID, bits [24:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

When the virtual machine writes to this register, the specified interrupt in the List registers is changed from active to inactive, or from active and pending to pending. If the specified interrupt is present in the List registers but is not in either the active or active and pending states, the effect is UNPREDICTABLE. If the specified interrupt is not present in the List registers, GIC_HCR.EOIcount is incremented, potentially generating a maintenance interrupt.

**Note**

If the specified interrupt is not present in the List registers, the virtual machine cannot recover the INTID. Therefore, the hypervisor must ensure that, when GICV_CTLR.EOImode == 1, no more than one active interrupt is transferred from the List registers into a software list. If more than one active
interrupt that is not stored in the List registers exists, the hypervisor must handle accesses to GICV_DIR in software, typically by trapping these accesses.

If the corresponding GICH_LR<n>.HW == 1, indicating a hardware interrupt, then a deactivate request is sent to the physical Distributor, identifying the physical INTID from the corresponding field in the List register. This effect is identical to a Non-secure write to GICC_DIR from the PE having that physical INTID. This means that if the corresponding physical interrupt is marked as Group 0, the request is ignored.

**Note**

Interrupt deactivation using this register is based on the provided INTID, with no requirement to deactivate interrupts in any particular order. A single register is therefore used to deactivate both Group 0 and Group 1 interrupts.

**Accessing the GICV_DIR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_DIR provides equivalent functionality.
- For AArch64 implementations, ICC_DIR_EL1 provides equivalent functionality.

Writes to this register are valid only when GICV_CTLR.EOImode == 1. Writes to this register are otherwise UNPREDICTABLE.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

**GICV_DIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x1000</td>
<td>GICV_DIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
GICV_EOIR, Virtual Machine End Of Interrupt Register

The GICV_EOIR characteristics are:

Purpose

A write to this register performs a priority drop for the specified Group 0 virtual interrupt and, if GICV_CTLR.EOImode == 0, also deactivates the interrupt.

This register corresponds to the physical CPU interface register GICC_EOIR.

Configuration

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_EOIR is a 32-bit register.

Field descriptions

The GICV_EOIR bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0 | INTID |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

The behavior of this register depends on the setting of GICV_CTLR.EOImode:

<table>
<thead>
<tr>
<th>GICV_CTLR.EOImode</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Both the priority drop and the deactivate interrupt effects occur</td>
</tr>
<tr>
<td>0b1</td>
<td>Only the priority drop effect occurs.</td>
</tr>
</tbody>
</table>

A successful EOI request means that:

- The highest priority bit in GICH_APR<n> is cleared, causing the running priority to drop.
- If the appropriate GICV_CTLR.EOImode bit == 0, the interrupt is deactivated in the corresponding List register GICH_LR<n>. If GICH_LR<n>.HW == 1, indicating the INTID corresponds to a hardware interrupt, a deactivate request is also sent to the physical Distributor, identifying the physical INTID from the
corresponding field in the List register. This effect is identical to a Non-secure write to GICC_DIR from the PE having that physical INTID. This means that if the corresponding physical interrupt is marked as Group 0, and GICD_CTLR.DS == 0, the deactivation request is ignored. See GICC_EOIR for more information.

**Note**

Only Group 1 interrupts can target the hypervisor, and therefore only Group 1 interrupts are deactivated in the Distributor.

### Accessing the GICV_EOIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_EOIR0 provides equivalent functionality.
- For AArch64 implementations, ICC_EOIR0_EL1 provides equivalent functionality.

This register is used for Group 0 interrupts only. GICV_AEOIR provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

**GICV_EOIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x0010</td>
<td>GICV_EOIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
GICV_HPPIR, Virtual Machine Highest Priority Pending Interrupt Register

The GICV_HPPIR characteristics are:

**Purpose**

Provides the INTID of the highest priority pending Group 0 virtual interrupt in the List registers.

This register corresponds to the physical CPU interface register GICC_HPPIR.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_HPPIR is a 32-bit register.

**Field descriptions**

The GICV_HPPIR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| INTID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

**Bits [31:25]**

Reserved, RES0.

**INTID, bits [24:0]**

The INTID of the signaled interrupt.

---

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

---

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Reads of the GICC_HPPIR that do not return a valid INTID return a spurious INTID, 1022 or 1023. See Special INTIDs.


<table>
<thead>
<tr>
<th>Highest priority pending interrupt Group</th>
<th>GICV_HPPIR read</th>
<th>GICV_CTRLR.AckCtl</th>
<th>Returned INTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Non-secure</td>
<td>x</td>
<td>1022</td>
</tr>
<tr>
<td>1</td>
<td>Secure</td>
<td>0</td>
<td>1023</td>
</tr>
<tr>
<td>0</td>
<td>Non-secure</td>
<td>x</td>
<td>1023</td>
</tr>
<tr>
<td>0</td>
<td>Secure</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>No pending interrupts</td>
<td>x</td>
<td>x</td>
<td>1023</td>
</tr>
</tbody>
</table>

If the CPU interface supports only a single Security state, the entries that apply to Secure reads describe the behavior.

### Accessing the GICV_HPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_HPPIR0 provides equivalent functionality.
- For AArch64 implementations, ICC_HPPIR0_EL1 provides equivalent functionality.

This register is used for Group 0 interrupts only. GICV_AHPPIR provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

**GICV_HPPIR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x0018</td>
<td>GICV_HPPIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.
The GICV_IAR characteristics are:

**Purpose**

Provides the INTID of the signaled Group 0 virtual interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

This register corresponds to the physical CPU interface register GICC_IAR.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_IAR is a 32-bit register.

**Field descriptions**

The GICV_IAR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>INTID</td>
</tr>
</tbody>
</table>

**Bits [31:25]**

Reserved, RES0.

**INTID, bits [24:0]**

The INTID of the signaled interrupt.

**Note**

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

When the virtual machine writes to this register, the virtual CPU interface acknowledges the highest priority pending virtual interrupt and sets the state in the corresponding List register to active. The appropriate bit in the active priorities register GICH_APR<\n> is set to 1.

If GICH_LR<\n>.HW == 0, indicating that the interrupt is software-triggered, then bits [12:10] of GICH_LR<\n> are returned in bits [12:10] of GICV_IAR. Otherwise bits [12:10] are RES0.

A read of this register returns the spurious INTID 1023 if either of the following is true:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE with the virtual CPU interface enabled and GICH_HCR.En == 1.
Interrupt signaling by the virtual CPU interface is disabled.

A read of this register returns the spurious INTID 1022 if the highest priority pending interrupt is Group 1 and GICV_CTLR.AckCtl == 0.

Accessing the GICV_IAR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_IAR0 provides equivalent functionality.
- For AArch64 implementations, ICC_IAR0_EL1 provides equivalent functionality.

This register is used for Group 0 interrupts only. GICV_AIAR provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_IAR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x000C</td>
<td>GICV_IAR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GICV_IIDR, Virtual Machine CPU Interface Identification Register

The GICV_IIDR characteristics are:

**Purpose**

Provides information about the implementer and revision of the virtual CPU interface.

**Configuration**

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

**Attributes**

GICV_IIDR is a 32-bit register.

**Field descriptions**

The GICV_IIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>ProductID</th>
<th>Architecture_version</th>
<th>Revision</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-20</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**ProductID, bits [31:20]**

An *IMPLEMENTATION DEFINED* product identifier.

**Architecture_version, bits [19:16]**

The version of the GIC architecture that is implemented.

<table>
<thead>
<tr>
<th>Architecture_version</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>GICv1.</td>
</tr>
<tr>
<td>0b0010</td>
<td>GICv2.</td>
</tr>
<tr>
<td>0b0011</td>
<td>GICv3 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.</td>
</tr>
<tr>
<td>0b0100</td>
<td>GICv4 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1.</td>
</tr>
</tbody>
</table>

Other values are reserved.

**Revision, bits [15:12]**

An *IMPLEMENTATION DEFINED* revision number for the CPU interface.

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the CPU interface.

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

**Accessing the GICV_IIDR**

**GICV_IIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x00FC</td>
<td>GICV_IIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
- When GICD_CTLR.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The GICV_PMR characteristics are:

**Purpose**

This register provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

**Note**

Higher interrupt priority corresponds to a lower value of the Priority field.

This register corresponds to the physical CPU interface register GICC_PMR.

**Configuration**

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is available when the GIC implementation supports interrupt virtualization.

The Priority field of this register is aliased to GICH_VMCR.VMPR, to enable state to be switched easily between virtual machines during context-switching.

**Attributes**

GICV_PMR is a 32-bit register.

**Field descriptions**

The GICV_PMR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>Priority</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The priority mask level for the virtual CPU interface. If the priority of the interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

If the GIC implementation supports fewer than 256 priority levels some bits might be RAZ/WI, as follows:

- For 128 supported levels, bit [0] = 0b0.
- For 64 supported levels, bits [1:0] = 0b00.
- For 32 supported levels, bits [2:0] = 0b000.
- For 16 supported levels, bits [3:0] = 0b0000.

See Interrupt prioritization, section 4.8 of the GICv3 Architecture Specification for more information.

This field resets to an architecturally **UNKNOWN** value.
Accessing the GICV_PMR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_PMR provides equivalent functionality.
- For AArch64 implementations, ICC_PMR_EL1 provides equivalent functionality.

GICV_PMR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU</td>
<td>0x0004</td>
<td>GICV_PMR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GICV_RPR, Virtual Machine Running Priority Register

The GICV_RPR characteristics are:

**Purpose**

This register indicates the running priority of the virtual CPU interface.

This register corresponds to the physical CPU interface register GICC_RPR.

**Configuration**

This register is available when the GIC implementation supports interrupt virtualization.

**Attributes**

GICV_RPR is a 32-bit register.

**Field descriptions**

The GICV_RPR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | Priority |

**Bits [31:8]**

Reserved, RES0.

**Priority, bits [7:0]**

The current running priority on the virtual CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR was set to the minimum value.

**Accessing the GICV_RPR**

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, ICC_RPR provides equivalent functionality.
- For AArch64 implementations, ICC_RPR_EL1 provides equivalent functionality.

Depending on the implementation, if no bits are set to 1 in GICH_APR<n>, indicating no active virtual interrupts in the virtual CPU interface, the priority reads as 0xFF or 0xF8 to reflect the number of supported interrupt priority bits defined by GICH_VTR_PRIbits.

**GICV_RPR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x0014</td>
<td>GICV_RPR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When GICD_CTLR.DS == 0b0 accesses to this register are \textbf{RO}.
• When IsAccessSecure() accesses to this register are \textbf{RO}.
• When \texttt{!IsAccessSecure()} accesses to this register are \textbf{RO}.
GICV_STATUSR, Virtual Machine Error Reporting Status Register

The GICV_STATUSR characteristics are:

**Purpose**

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

**Configuration**

In systems where this register is implemented, Arm expects that when a virtual machine is scheduled, the hypervisor ensures that this register is cleared to 0. The hypervisor might check for illegal accesses when the virtual machine is unscheduled.

**Attributes**

GICV_STATUSR is a 32-bit register.

**Field descriptions**

The GICV_STATUSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**WROD, bit [3]**

<table>
<thead>
<tr>
<th>WROD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>01</td>
<td>A write to an RO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

**RWOD, bit [2]**

<table>
<thead>
<tr>
<th>RWOD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>01</td>
<td>A read of a WO location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.
WRD, bit [1]

Write to a reserved location.

<table>
<thead>
<tr>
<th>WRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A write to a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

<table>
<thead>
<tr>
<th>RRD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>A read of a reserved location has been detected.</td>
</tr>
</tbody>
</table>

When a violation is detected, software must write 1 to this register to reset it.

Accessing the GICV_STATUSR

This is an optional register. If the register is implemented, GICC_STATUSR must also be implemented. If the register is not implemented, the location is RAZ/WI.

This register is used only when System register access is not enabled. If System register access is enabled, this register is not updated. Equivalent function might be provided by appropriate traps and exceptions.

GICV_STATUSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC Virtual CPU interface</td>
<td>0x002C</td>
<td>GICV_STATUSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
The GITS_BASER<n> characteristics are:

**Purpose**

Specifies the base address and size of the ITS translation tables.

**Configuration**

Some or all RW fields of this register have defined reset values.

A copy of this register is provided for each ITS translation table.

Bits [63:32] and bits [31:0] are accessible independently.

A maximum of 8 GITS_BASER<n> registers can be provided. Unimplemented registers are RES0.

When GITS_CTLR Enabled == 1 or GITS_CTLR Quiescent == 0, writing this register is UNPREDICTABLE.

**Attributes**

GITS_BASER<n> is a 64-bit register.

**Field descriptions**

The GITS_BASER<n> bit assignments are:

<p>|       | 63  | 62  | 61  | 60  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  | 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|</p>
<table>
<thead>
<tr>
<th>Valid</th>
<th>Indirect</th>
<th>InnerCache</th>
<th>Type</th>
<th>OuterCache</th>
<th>Entry Size</th>
<th>Physical Address</th>
<th>Shareability</th>
<th>Page Size</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
</tr>
</tbody>
</table>

**Valid, bit [63]**

Indicates whether software has allocated memory for the translation table:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No memory is allocated for the translation table. The ITS discards any writes to the interrupt translation page when either:</td>
</tr>
<tr>
<td></td>
<td>• GITS_BASER&lt;n&gt;.Type specifies any valid table entry type other than interrupt collections, that is, any value other than 0b100.</td>
</tr>
<tr>
<td></td>
<td>• GITS_BASER&lt;n&gt;.Type specifies an interrupt collection and GITS_TYPER.HCC == 0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is allocated to the translation table.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Indirect, bit [62]**

This field indicates whether an implemented register specifies a single, flat table or a two-level table where the first level contains a list of descriptors.
Indirect

<table>
<thead>
<tr>
<th>Indirect</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Single Level. The Size field indicates the number of pages used by the ITS to store data associated with each table entry.</td>
</tr>
<tr>
<td>0b1</td>
<td>Two Level. The Size field indicates the number of pages which contain an array of 64-bit descriptors to pages that are used to store the data associated with each table entry. A little endian memory order model is used.</td>
</tr>
</tbody>
</table>

See The ITS tables for more information.

This field is RAZ/WI for GIC implementations that only support flat tables. If the maximum width of the scaling factor that is identified by GITS_BASER<n>.Type and the smallest page size that is supported result in a single level table that requires multiple pages, then implementing this bit as RAZ/WI is DEPRECATED.

This field resets to an architecturally UNKNOWN value.

**InnerCache, bits [61:59]**

Indicates the Inner Cacheability attributes of accesses to the table. The possible values of this field are:

<table>
<thead>
<tr>
<th>InnerCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Inner Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Inner Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Inner Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Inner Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally UNKNOWN value.

**Type, bits [58:56]**

Read only. Specifies the type of entity that requires entries in the corresponding translation table. The possible values of the field are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Unimplemented. This register does not correspond to a translation table.</td>
</tr>
<tr>
<td>0b001</td>
<td>Devices. This register corresponds to a translation table that scales with the width of the DeviceID. Only a single GITS_BASER&lt;n&gt; register reports this type.</td>
</tr>
<tr>
<td>0b010</td>
<td>vPEs. GICv4 only. This register corresponds to a translation table that scales with the number of vPEs in the system. The translation table requires (ENTRY_SIZE * N) bytes of memory, where N is the number of vPEs in the system. Only a single GITS_BASER&lt;n&gt; register reports this type.</td>
</tr>
<tr>
<td>0b100</td>
<td>Interrupt collections. This register corresponds to a translation table that scales with the number of interrupt collections in the system. The translation table requires (ENTRY_SIZE * N) bytes of memory, where N is the number of interrupt collections. Not more than one GITS_BASER&lt;n&gt; register will report this type.</td>
</tr>
</tbody>
</table>

Other values are reserved.

**Note**

The minimum number of entries that an ITS must support is N+1, where N is the number of physical PEs in the system.

Arm recommends that the GITS_BASER<n> registers are allocated as follows:

- GITS_BASER0.Type is 0b001 (Device).
• GITS_BASER1.Type is either 0b100 (Collection Table) or 0b000 (Unimplemented).

• GITS_BASER2.Type is either 0b010 (vPE) or 0b000 (Unimplemented).

• GITS_BASER<n>.Type, where ‘n’ is in the range 3 to 7, is 0b000 (Unimplemented).

Other allocations of Type values are deprecated.

This field resets to an architecturally UNKNOWN value.

**OuterCache, bits [55:53]**

Indicates the Outer Cacheability attributes of accesses to the table. The possible values of this field are:

<table>
<thead>
<tr>
<th>OuterCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Outer Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Outer Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Outer Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Outer Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

**Entry_Size, bits [52:48]**

Read-only. Specifies the number of bytes per translation table entry, minus one.

**Physical_Address, bits [47:12]**

Physical Address. When Page_Size is 4KB or 16KB:

• Bits [51:48] of the base physical address are zero.
• This field provides bits[47:12] of the base physical address of the table.
• Bits[11:0] of the base physical address are zero.
• The address must be aligned to the size specified in the Page Size field. Otherwise the effect is CONSTRAINED UNPREDICTABLE, and can be one of the following:
  ◦ Bits[X:12], where X is derived from the page size, are treated as zero.
  ◦ The value of bits[X:12] are used when calculating the address of a table access.

When Page_Size is 64KB:

• Bits[47:16] of the register provide bits[47:16] of the base physical address of the table.
• Bits[15:12] of the register provide bits[51:48] of the base physical address of the table.
• Bits[15:0] of the base physical address are 0.

In implementations that support fewer than 52 bits of physical address, any unimplemented upper bits might be RAZ/WI.

This field resets to an architecturally UNKNOWN value.

**Shareability, bits [11:10]**

Indicates the Shareability attributes of accesses to the table. The possible values of this field are:
It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

**Page_Size, bits [9:8]**

The size of page that the translation table uses:

<table>
<thead>
<tr>
<th>Page_Size</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>4KB.</td>
</tr>
<tr>
<td>0b01</td>
<td>16KB.</td>
</tr>
<tr>
<td>0b10</td>
<td>64KB.</td>
</tr>
<tr>
<td>0b11</td>
<td>Reserved. Treated as 0b10.</td>
</tr>
</tbody>
</table>

Note

If the GIC implementation supports only a single, fixed page size, this field might be RO.

This field resets to an architecturally UNKNOWN value.

**Size, bits [7:0]**

The number of pages of physical memory allocated to the table, minus one. GITS_BASER<n>.Page_Size specifies the size of each page.

If GITS_BASER<n>.Type == 0, this field is RAZ/WI.

This field resets to an architecturally UNKNOWN value.

**Accessing the GITS_BASER<n>**

GITS_BASER<n> can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0100 + 8n</td>
<td>GITS_BASER&lt;n&gt;</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRL.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
The GITS_CBASER characteristics are:

**Purpose**

Specifies the base address and size of the ITS command queue.

**Configuration**

Some or all RW fields of this register have defined reset values.

Bits [63:32] and bits [31:0] are accessible separately.

**Attributes**

GITS_CBASER is a 64-bit register.

**Field descriptions**

The GITS_CBASER bit assignments are:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
<th>54</th>
<th>53</th>
<th>52</th>
<th>51</th>
<th>50</th>
<th>49</th>
<th>48</th>
<th>47</th>
<th>46</th>
<th>45</th>
<th>44</th>
<th>43</th>
<th>42</th>
<th>41</th>
<th>40</th>
<th>39</th>
<th>38</th>
<th>37</th>
<th>36</th>
<th>35</th>
<th>34</th>
<th>33</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>RES0</td>
<td>InnerCache</td>
<td>RES0</td>
<td>OuterCache</td>
<td>RES0</td>
<td>Physical_Address</td>
<td>Physical_Address</td>
<td>Shareability</td>
<td>RES0</td>
<td>Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Valid, bit [63]**

Indicates whether software has allocated memory for the command queue:

<table>
<thead>
<tr>
<th>Valid</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No memory is allocated for the command queue.</td>
</tr>
<tr>
<td>0b1</td>
<td>Memory is allocated to the command queue.</td>
</tr>
</tbody>
</table>

This field resets to 0.

**Bit [62]**

Reserved, RES0.

**InnerCache, bits [61:59]**

Indicates the Inner Cacheability attributes of accesses to the command queue. The possible values of this field are:

<table>
<thead>
<tr>
<th>InnerCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Device-nGnRnE.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Inner Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Inner Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Inner Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Inner Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Inner Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Inner Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

This field resets to an architecturally **UNKNOWN** value.
Bits [58:56]

Reserved, RES0.

OuterCache, bits [55:53]

Indicates the Outer Cacheability attributes of accesses to the command queue. The possible values of this field are:

<table>
<thead>
<tr>
<th>OuterCache</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability.</td>
</tr>
<tr>
<td>0b001</td>
<td>Normal Outer Non-cacheable.</td>
</tr>
<tr>
<td>0b010</td>
<td>Normal Outer Cacheable Read-allocate, Write-through.</td>
</tr>
<tr>
<td>0b011</td>
<td>Normal Outer Cacheable Read-allocate, Write-back.</td>
</tr>
<tr>
<td>0b100</td>
<td>Normal Outer Cacheable Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b101</td>
<td>Normal Outer Cacheable Write-allocate, Write-back.</td>
</tr>
<tr>
<td>0b110</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-through.</td>
</tr>
<tr>
<td>0b111</td>
<td>Normal Outer Cacheable Read-allocate, Write-allocate, Write-back.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

Bit [52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the base physical address of the command queue. Bits [11:0] of the base address are 0.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

If bits [15:12] are not all zeros, behavior is a CONSTRAINED UNPREDICTABLE choice:

- Bits [15:12] are treated as if all the bits are zero. The value read back from those bits is either the value written or zero.
- The result of the calculation of an address for a command queue read can be corrupted.

This field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the command queue. The possible values of this field are:

<table>
<thead>
<tr>
<th>Shareability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-shareable.</td>
</tr>
<tr>
<td>0b01</td>
<td>Inner Shareable.</td>
</tr>
<tr>
<td>0b10</td>
<td>Outer Shareable.</td>
</tr>
<tr>
<td>0b11</td>
<td>Reserved. Treated as 0b00.</td>
</tr>
</tbody>
</table>

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

This field resets to an architecturally UNKNOWN value.

Bits [9:8]

Reserved, RES0.
Size, bits [7:0]

The number of 4KB pages of physical memory allocated to the command queue, minus one.

This field resets to an architecturally UNKNOWN value.

The command queue is a circular buffer and wraps at Physical Address [47:0] + (4096 * (Size + 1)).

Note

When this register is successfully written, the value of GITS_CREADR is set to zero.

Accessing the GITS_CBASER

When GITS_CTLR. Enabled == 1 or GITS_CTLR. Quiescent == 0, writing this register is UNPREDICTABLE.

GITS_CBASER can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0080</td>
<td>GITS_CBASER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GITS_CREADR, ITS Read Register

The GITS_CREADR characteristics are:

Purpose

Specifies the offset from GITS_CBASER where the ITS reads the next ITS command.

Configuration

This register is cleared to 0 when a value is written to GITS_CBASER.

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CREADR is a 64-bit register.

Field descriptions

The GITS_CREADR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    | RES0 |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

RES0

Offset

Stalled

Bits [63:20]  
Reserved, RES0.

Offset, bits [19:5]


Bits [4:1]

Reserved, RES0.

Stalled, bit [0]

Reports whether the processing of commands is stalled because of a command error.

<table>
<thead>
<tr>
<th>Stalled</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ITS command queue is not stalled because of a command error.</td>
</tr>
<tr>
<td>0b1</td>
<td>ITS command queue is stalled because of a command error.</td>
</tr>
</tbody>
</table>

See The ITS Command Interface for more information.

Accessing the GITS_CREADR

GITS_CREADR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RO**.
- When IsAccessSecure() accesses to this register are **RO**.
- When !IsAccessSecure() accesses to this register are **RO**.
GITS_CTLR, ITS Control Register

The GITS_CTLR characteristics are:

**Purpose**

Controls the operation of an ITS.

**Configuration**

Some or all RW fields of this register have defined reset values.

The ITS_Number (bits [7:4]) and bit [1] fields apply only in GICv4 implementations, and are RES0 in GICv3 implementations.

**Attributes**

GITS_CTLR is a 32-bit register.

**Field descriptions**

The GITS_CTLR bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Quiescent | RES0 | ITS_Number | RES0 | ImDe | Enabled |

**Quiescent, bit [31]**

Read-only. Indicates completion of all ITS operations when GITS_CTLR.Enabled == 0.

<table>
<thead>
<tr>
<th>Quiescent</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The ITS is not quiescent and cannot be powered down.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ITS is quiescent and can be powered down.</td>
</tr>
</tbody>
</table>

For the ITS to be quiescent, there must be no transactions in progress. In addition, all operations required to ensure that mapping data is consistent with external memory must be complete.

**Note**

In distributed GIC implementations, this bit is set to 1 only after the ITS forwards any operations that have not yet been completed to the Redistributors and receives confirmation that all such operations have reached the appropriate Distributor.

When GITS_CTLR.Enabled==1 the value of GITS_CTLR.Quiescent is UNKNOWN.

This field resets to 1.

**Bits [30:8]**

Reserved, RES0.

**ITS_Number, bits [7:4]**

In GICv3 implementations this is RES0.

In GICv4 implementations with more than one ITS instance, this field indicates the ITS number for use with VMOVP.
It is **IMPLEMENTATION DEFINED** whether this field is programmable or RO.

If this field is programmable, changing this field when GITS_CTLR.Quiescent==0 or GITS_CTLR.Enabled==1 is **UNPREDICTABLE**.

This field resets to an architecturally **UNKNOWN** value.

**Bits [3:2]**

Reserved, RES0.

**ImDe, bit [1]**

In GICv3 implementations this bit is RES0.

In GICv4 implementations this bit is **IMPLEMENTATION DEFINED**.

This field resets to 0.

**Enabled, bit [0]**

Controls whether the ITS is enabled:

<table>
<thead>
<tr>
<th>Enabled</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The ITS is not enabled. Writes to GITS_TRANSLATER are ignored and no further command queue entries are processed.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ITS is enabled. Writes to GITS_TRANSLATER result in interrupt translations and the command queue is processed.</td>
</tr>
</tbody>
</table>

If a write to this register changes this field from 1 to 0, the ITS must ensure that both:

- Any caches containing mapping data are made consistent with external memory.
- GITS_CTLR.Quiescent == 0 until all caches are consistent with external memory.

Changing GITS_CTLR.Enabled from 0 to 1 when GITS_CTLR.Quiescent is 0 results in **UNPREDICTABLE** behavior.

This field resets to 0.

**Accessing the GITS_CTLR**

**GITS_CTLR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0000</td>
<td>GITS_CTLR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
- When !IsAccessSecure() accesses to this register are **RW**.
GITS_CWRITER, ITS Write Register

The GITS_CWRITER characteristics are:

**Purpose**

Specifies the offset from GITS_CBASER where software writes the next ITS command.

**Configuration**

RW fields in this register reset to architecturally UNKNOWN values.

Bits [63:32] and bits [31:0] are accessible separately.

**Attributes**

GITS_CWRITER is a 64-bit register.

**Field descriptions**

The GITS_CWRITER bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>62</td>
<td>Offset, bits [19:5]</td>
</tr>
<tr>
<td>61</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>60</td>
<td>Retry, bit [0]</td>
</tr>
<tr>
<td>59</td>
<td>Bits [4:1]</td>
</tr>
<tr>
<td>58</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>57</td>
<td>Bits [63:20]</td>
</tr>
<tr>
<td>56</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>55</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>54</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>53</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>52</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>51</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>50</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>49</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>48</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>47</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>46</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>45</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>44</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>43</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>42</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>41</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>40</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>39</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>38</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>37</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>36</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>35</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>34</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>33</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>32</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Bits [63:20]**

Reserved, RES0.

**Offset, bits [19:5]**


This field resets to an architecturally UNKNOWN value.

**Bits [4:1]**

Reserved, RES0.

**Retry, bit [0]**

Writing this bit has the following effects:

<table>
<thead>
<tr>
<th>Retry</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No effect on the processing commands by the ITS.</td>
</tr>
<tr>
<td>0b1</td>
<td>Restart the processing of commands by the ITS if it stalled because of a command error.</td>
</tr>
</tbody>
</table>

**Note**

If the processing of commands is not stalled because of a command error, writing 1 to this bit has no effect.

When read, this bit is RES0.
See The ITS Command Interface for more information.

If GITS_CWRITER is written with a value outside of the valid range specified by GITS_CBASER.Physical_Address and GITS_CBASER.Size, behavior is a CONSTRAINED UNPREDICTABLE choice, as follows:

- The command queue is considered invalid, and no further commands are processed until GITS_CWRITER is written with a value that is in the valid range.
- The value is treated as a valid UNKNOWN value.

An implementation might choose to report a system error in an IMPLEMENTATION DEFINED manner.

Accessing the GITS_CWRITER

GITS_CWRITER can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0088</td>
<td>GITS_CWRITER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RW.
- When IsAccessSecure() accesses to this register are RW.
- When !IsAccessSecure() accesses to this register are RW.
GITS_IIDR, ITS Identification Register

The GITS_IIDR characteristics are:

**Purpose**

Provides information about the implementer and revision of the ITS.

**Configuration**

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

**Attributes**

GITS_IIDR is a 32-bit register.

**Field descriptions**

The GITS_IIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>ProductID</td>
</tr>
<tr>
<td>23-20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19-16</td>
<td>Variant</td>
</tr>
<tr>
<td>15-12</td>
<td>Revision</td>
</tr>
<tr>
<td>11-0</td>
<td>Implementer</td>
</tr>
</tbody>
</table>

**ProductID, bits [31:24]**

An IMPLEMENTATION DEFINED product identifier.

**Bits [23:20]**

Reserved, RES0.

**Variant, bits [19:16]**

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

**Revision, bits [15:12]**

An IMPLEMENTATION DEFINED revision number. Typically, this field is used to distinguish minor revisions of a product.

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the ITS:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.
Accessing the GITS_IIDR

GITS_IIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0004</td>
<td>GITS_IIDR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GITS_MPAMIDR, Report maximum PARTID and PMG Register

The GITS_MPAMIDR characteristics are:

**Purpose**

Reports the maximum support PARTID and PMG values.

**Configuration**

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GITS_MPAMIDR are RES0.

A copy of this register is provided for each ITS.

When GITS_TYPER.MPAM==0, this register is RES0.

**Attributes**

GITS_MPAMIDR is a 32-bit register.

**Field descriptions**

The GITS_MPAMIDR bit assignments are:

| 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 |
|---------------------------------|---------------------------------|-------------------------------|
| RES0   PMGmax   PARTIDmax         |

**Bits [31:24]**

Reserved, RES0.

**PMGmax, bits [23:16]**

Maximum PMG value supported.

**PARTIDmax, bits [15:0]**

Maximum PARTID value supported.

**Accessing the GITS_MPAMIDR**

GITS_MPAMIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0010</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
GITS_PARTIDR, Set PARTID and PMG Register

The GITS_PARTIDR characteristics are:

**Purpose**

Sets the PARTID and PMG values used for memory accesses by the ITS.

**Configuration**

This register is present only when GIC, >=3.1 is implemented. Otherwise, direct accesses to GITS_PARTIDR are RES0. A copy of this register is provided for each ITS.

When GITS_TYPER.MPAM==0, this register is RES0.

**Attributes**

GITS_PARTIDR is a 32-bit register.

**Field descriptions**

The GITS_PARTIDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | PMG | PARTID |

**Bits [31:24]**

Reserved, RES0.

**PMG, bits [23:16]**

PMG value used when ITS accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PMG values in the range 0 to PMG_MAX are stateful or RES0.

**PARTID, bits [15:0]**

PARTID value used when ITS accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PARTID values in the range 0 to PARTID_MAX are stateful or RES0.

**Accessing the GITS_PARTIDR**

GITS_PARTIDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0014</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTRLR.DS == 0b0 accesses to this register are **RW**.
- When IsAccessSecure() accesses to this register are **RW**.
• When !IsAccessSecure() accesses to this register are **RW**.
GITS_TRANSLATER, ITS Translation Register

The GITS_TRANSLATER characteristics are:

**Purpose**

Written by a requesting Device to signal an interrupt for translation by the ITS.

**Configuration**

This register is at the same offset as **GICD_SETSPI_NSR** in the Distributor, and is at the same offset as **GICR_SETLPIR** in the Redistributor.

**Attributes**

GITS_TRANSLATER is a 32-bit register.

**Field descriptions**

The GITS_TRANSLATER bit assignments are:

```
 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
```

**EventID, bits [31:0]**

An identifier corresponding to the interrupt to be translated.

**Note**

The size of the EventID is DeviceID specific, and set when the DeviceID is mapped to an ITT (using **MAPD**).

The number of EventID bits implemented is reported by **GITS_TYPER_ID_bits**. If a write specifies non-zero identifiers bits outside this range behavior is a **CONSTRAINED UNPREDICTABLE** choice between:

- Non-zero identifier bits outside the supported range are ignored.
- The write is ignored.

The DeviceID presented to an ITS is used to index a device table. The device table maps the DeviceID to an interrupt translation table for that device.

**Accessing the GITS_TRANSLATER**

16-bit access to bits [15:0] of this register must be supported. When this register is written by a 16-bit transaction, bits [31:16] are written as zero.

Implementations must ensure that:

- A unique DeviceID is provided for each requesting device, and the DeviceID is presented to the ITS when a write to this register occurs in a manner that cannot be spoofed by any agent capable of performing writes.
- The DeviceID presented corresponds to the DeviceID field in the ITS commands.

Writes to this register are ignored if any of the following are true:

- **GITS_CTLR**.Enabled == 0.
- The presented DeviceID is not mapped to an Interrupt Translation Table.
- The DeviceID is larger than the supported size.
The DeviceID is mapped to an Interrupt Translation Table, but the EventID is outside the range specified by MAPD.

The EventID is mapped to an Interrupt Translation Table and the EventID is within the range specified by MAPD, but the EventID is unmapped.

Translation requests that result from writes to this register are subject to certain ordering rules. See Ordering of translations following writes to GITS_TRANSLATER for more information.

**GITS_TRANSLATER can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS translation</td>
<td>0x0040</td>
<td>GITS_TRANSLATER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are **WO**.
- When IsAccessSecure() accesses to this register are **WO**.
- When !IsAccessSecure() accesses to this register are **WO**.
GITS_TYPER, ITS Type Register

The GITS_TYPER characteristics are:

**Purpose**

Specifies the features that an ITS supports.

**Configuration**

**Attributes**

GITS_TYPER is a 64-bit register.

**Field descriptions**

The GITS_TYPER bit assignments are:

<table>
<thead>
<tr>
<th>Bits [63:39]</th>
<th>Reserved, RES0.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MPAM, bit [38]</strong> When GIC, &gt;=3.1 is implemented:</td>
<td></td>
</tr>
<tr>
<td>MPAM</td>
<td>Meaning</td>
</tr>
<tr>
<td>0b0</td>
<td>MPAM not supported.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM supported.</td>
</tr>
<tr>
<td>Otherwise:</td>
<td></td>
</tr>
<tr>
<td>Reserved, RES0.</td>
<td></td>
</tr>
<tr>
<td><strong>VMOVP, bit [37]</strong> Indicates the form of the VMOVP command.</td>
<td></td>
</tr>
<tr>
<td>VMOVP</td>
<td>Meaning</td>
</tr>
<tr>
<td>0b0</td>
<td>When moving a vPE, software must issue a VMOVP on all ITSs that have mappings for that vPE. The ITSList and Sequence Number fields in the VMOVP command must ensure synchronization, otherwise behavior is UNPREDICTABLE.</td>
</tr>
<tr>
<td>0b1</td>
<td>When moving a vPE, software must only issue a VMOVP on one of the ITSs that has a mapping for that vPE. The ITSList and Sequence Number fields in the VMOVP command are RES0.</td>
</tr>
</tbody>
</table>
CIL, bit [36]

Collection ID Limit.

<table>
<thead>
<tr>
<th>CIL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ITS supports 16-bit Collection ID. GITS_TYPER.CIDbits is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>GITS_TYPER.CIDbits indicates supported Collection ID size</td>
</tr>
</tbody>
</table>

In implementations that do not support Collections in external memory, this bit is RES0 and the number of Collections supported is reported by GITS_TYPER.HCC.

CIDbits, bits [35:32]

Number of Collection ID bits.

- The number of bits of Collection ID - 1.
- When GITS_TYPER.CIL==0, this field is RES0.

HCC, bits [31:24]

Hardware Collection Count. The number of interrupt collections supported by the ITS without provisioning of external memory.

---

Note

Collections held in hardware are unmapped at reset.

Bits [23:20]

Reserved, RES0.

PTA, bit [19]

Physical Target Addresses. Indicates the format of the target address:

<table>
<thead>
<tr>
<th>PTA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The target address corresponds to the PE number specified by GICR_TYPER.Processor_Number.</td>
</tr>
<tr>
<td>0b1</td>
<td>The target address corresponds to the base physical address of the required Redistributor.</td>
</tr>
</tbody>
</table>

See Target addresses for more information.

SEIS, bit [18]

SEI support. Indicates whether the virtual CPU interface supports generation of SEIs:

<table>
<thead>
<tr>
<th>SEIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The ITS does not support local generation of SEIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ITS supports local generation of SEIs.</td>
</tr>
</tbody>
</table>

Devbits, bits [17:13]

The number of DeviceID bits implemented, minus one.

ID_bits, bits [12:8]

The number of EventID bits implemented, minus one.
**ITT_entry_size, bits [7:4]**

Read-only. Indicates the number of bytes per translation table entry, minus one.

See the ITS command 'MAPD' for more information.

**IMPLEMENTATION DEFINED, bit [3]**

IMPLEMENTATION DEFINED.

**CCT, bit [2]**

Cumulative Collection Tables.

<table>
<thead>
<tr>
<th>CCT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The total number of supported collections is determined by the number of collections held in memory only.</td>
</tr>
<tr>
<td>0b1</td>
<td>The total number of supported collections is determined by number of collections that are held in memory and the number indicated by GITS_TYPER.HCC.</td>
</tr>
</tbody>
</table>

If GITS_TYPER.HCC==0, or if memory backed collections are not supported (all GITS_BASER<n>_Type != 100), this bit is RES0.

**Virtual, bit [1]**

Indicates whether the ITS supports virtual LPIs and direct injection of virtual LPIs:

<table>
<thead>
<tr>
<th>Virtual</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The ITS does not support virtual LPIs or direct injection of virtual LPIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ITS supports virtual LPIs and direct injection of virtual LPIs.</td>
</tr>
</tbody>
</table>

This field is RES0 in GICv3 implementations.

**Physical, bit [0]**

Indicates whether the ITS supports physical LPIs:

<table>
<thead>
<tr>
<th>Physical</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The ITS does not support physical LPIs.</td>
</tr>
<tr>
<td>0b1</td>
<td>The ITS supports physical LPIs.</td>
</tr>
</tbody>
</table>

This field is RES1, indicating that the ITS supports physical LPIs.

**Accessing the GITS_TYPER**

GITS_TYPER can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>GIC ITS control</td>
<td>0x0008</td>
<td>GITS_TYPER</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When GICD_CTLR.DS == 0b0 accesses to this register are RO.
- When IsAccessSecure() accesses to this register are RO.
- When !IsAccessSecure() accesses to this register are RO.
The MIDR_EL1 characteristics are:

**Purpose**

Provides identification information for the PE, including an implementer code for the device and a device ID number.

**Configuration**

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch64 System register \texttt{MIDR[31:0]}.

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register \texttt{MIDR[31:0]}.

It is \texttt{IMPLEMENTATION DEFINED} whether MIDR_EL1 is implemented in the Core power domain or in the Debug power domain.

**Attributes**

MIDR_EL1 is a 32-bit register.

**Field descriptions**

The MIDR_EL1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| Implementer | Variant | Architecture | PartNum | Revision |

**Implementer, bits [31:24]**

The Implementer bits [31:24] must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

<table>
<thead>
<tr>
<th>Hex representation</th>
<th>Implementer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Reserved for software use</td>
</tr>
<tr>
<td>0xC0</td>
<td>Ampere Computing</td>
</tr>
<tr>
<td>0x41</td>
<td>Arm Limited</td>
</tr>
<tr>
<td>0x42</td>
<td>Broadcom Corporation</td>
</tr>
<tr>
<td>0x43</td>
<td>Cavium Inc.</td>
</tr>
<tr>
<td>0x44</td>
<td>Digital Equipment Corporation</td>
</tr>
<tr>
<td>0x46</td>
<td>Fujitsu Ltd.</td>
</tr>
<tr>
<td>0x49</td>
<td>Infineon Technologies AG</td>
</tr>
<tr>
<td>0x40</td>
<td>Motorola or Freescale Semiconductor Inc.</td>
</tr>
<tr>
<td>0x4E</td>
<td>NVIDIA Corporation</td>
</tr>
<tr>
<td>0x58</td>
<td>Applied Micro Circuits Corporation</td>
</tr>
<tr>
<td>0x51</td>
<td>Qualcomm Inc.</td>
</tr>
<tr>
<td>0x56</td>
<td>Marvell International Ltd.</td>
</tr>
<tr>
<td>0x69</td>
<td>Intel Corporation</td>
</tr>
</tbody>
</table>

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

**Variant, bits [23:20]**

An \texttt{IMPLEMENTATION DEFINED} variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.
Architecture, bits [19:16]

The permitted values of this field are:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>Armv4.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Armv4T.</td>
</tr>
<tr>
<td>0b0011</td>
<td>Armv5 (obsolete).</td>
</tr>
<tr>
<td>0b0100</td>
<td>Armv5T.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Armv5TE.</td>
</tr>
<tr>
<td>0b0110</td>
<td>Armv5TEJ.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Armv6.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Architectural features are individually identified in the ID_* registers, see ID registers in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile</td>
</tr>
</tbody>
</table>

All other values are reserved.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

Accessing the MIDR_EL1

MIDR_EL1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0xD00</td>
<td>MIDR_EL1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are RO.
- Otherwise accesses to this register are IMPDEF.
The MPAMCFG_CMAX characteristics are:

**Purpose**

The MPAMCFG_CMAX is a 32-bit read-write register that controls the maximum fraction of the cache capacity that the PARTID selected by MPAMCFG_PART_SEL is permitted to allocate. MPAMCFG_CMAX_s controls cache maximum capacity for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CMAX_ns controls the cache maximum capacity for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

**Configuration**

The power domain of MPAMCFG_CMAX is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMCFG_CMAX are RES0.

**Attributes**

MPAMCFG_CMAX is a 32-bit register.

**Field descriptions**

The MPAMCFG_CMAX bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>CMAX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**CMAX, bits [15:0]**

Maximum cache capacity usage in fixed-point fraction format by the partition selected by MPAMCFG_PART_SEL. The fraction represents the portion of the total cache capacity that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_CCAP_IDR.CMAX WD. Unimplemented bits within the field are RAZ/WI. The implemented bits of the CMAX field are always the most-significant bits of the field.

The fixed-point fraction CMAX is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1 - 1/ w.

**Accessing the MPAMCFG_CMAX**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_CMAX_s must be accessible from the Secure MPAM feature page. MPAMCFG_CMAX_ns must be accessible from the Non-secure MPAM feature page.
MPAMCFG_CMAX_s and MPAMCFG_CMAX_ns must be separate registers. The Secure instance (MPAMCFG_CMAX_s) accesses the cache capacity partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_CMAX_ns) accesses the cache capacity partitioning used for Non-secure PARTIDs.

**MPAMCFG_CMAX can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0108</td>
<td>MPAMCFG_CMAX_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0108</td>
<td>MPAMCFG_CMAX_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MPAMCFG_CPBM characteristics are:

**Purpose**

The MPAMCFG_CPBM register is a read-write register that configures the cache portions that a PARTID is allowed to allocate. After setting MPAMCFG_PART_SEL with a PARTID, software (usually a hypervisor) writes to the MPAMCFG_CPBM register to configure which cache portions the PARTID is allowed to allocate.

MPAMCFG_CPBM_s controls cache portions for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_CPBM_ns controls the cache portions for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

**Configuration**

The power domain of MPAMCFG_CPBM is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_CPOR == 1. Otherwise, direct accesses to MPAMCFG_CPBM are RES0.

**Attributes**

MPAMCFG_CPBM is a 32768-bit register.

**Field descriptions**

The MPAMCFG_CPBM bit assignments are:

**CPBM<n>, bit [n], for n = 0 to 32767**

Each bit, CPBM<n>, grants permission to the PARTID to allocate cache lines within cache portion n.

<table>
<thead>
<tr>
<th>CPBM&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PARTID is not permitted to allocate into cache portion n.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PARTID is permitted to allocate within cache portion n.</td>
</tr>
</tbody>
</table>

The number of bits in the cache portion partitioning bit map of this component is given in MPAMF_CPOR_IDR.CPBM_WD. CPBM_WD contains a value from 1 to $2^{15}$, inclusive. Values of CPBM_WD greater than 32 require a group of 32-bit registers to access the CPBM, up to 1024 registers.

Bits CPBM<n>, where n is greater than CPBM_WD, are not required to be implemented.

**Accessing the MPAMCFG_CPBM**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_CPBM_s must be accessible from the Secure MPAM feature page. MPAMCFG_CPBM_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_CPBM_s and MPAMCFG_CPBM_ns must be separate registers. The Secure instance (MPAMCFG_CPBM_s) accesses the cache portion bitmaps used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_CPBM_ns) accesses the cache portion bitmaps used for Non-secure PARTIDs.
MPAMCFG_CPBM can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x1000</td>
<td>MPAMCFG_CPBM_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x1000</td>
<td>MPAMCFG_CPBM_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MPAMCFG_INTPARTID characteristics are:

**Purpose**

MPAMCFG_INTPARTID is a 32-bit read-write register that controls the mapping of the PARTID selected by MPAMCFG_PART_SEL into a narrower internal PARTID (intPARTID).

MPAMCFG_INTPARTID_s controls the mapping for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_INTPARTID_ns controls the mapping for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

The MPAMCFG_INTPARTID register associates the request PARTID (reqPARTID) in the MPAMCFG_PART_SEL register with an internal PARTID (intPARTID) in this register. To set that association, store reqPARTID into the MPAMCFG_PART_SEL register and then store the intPARTID into the MPAMCFG_INTPARTID register. To read the association, store reqPARTID into the MPAMCFG_PART_SEL register and then read MPAMCFG_INTPARTID.

If the intPARTID stored into MPAMCFG_INTPARTID is out-of-range or does not have the INTERNAL bit set, the association of reqPARTID to intPARTID is not written and MPAMF_ESR is set to indicate an intPARTID_Range error.

If MPAMCFG_PART_SEL.INTERNAL is 1 when MPAMCFG_INTPARTID is read or written, MPAMF_ESR is set to indicate an Unexpected_INTERNAL error.

**Configuration**

The power domain of MPAMCFG_INTPARTID is IMPLEMENTATION_DEFINED.

This register is present only when MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMCFG_INTPARTID are RES0.

**Attributes**

MPAMCFG_INTPARTID is a 32-bit register.

**Field descriptions**

The MPAMCFG_INTPARTID bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | INTERNAL | INTPARTID |

**Bits [31:17]**

Reserved, RES0.

**INTERNAL, bit [16]**

Internal PARTID flag.

This bit must be 1 when written to the register. If written as 0, the write will not update the reqPARTID to intPARTID association.

On a read of this register, the bit will always read the value last written.
**INTPARTID, bits [15:0]**

This field contains the intPARTID mapped to the reqPARTID in [MPAMCFG_PART_SEL](#). The maximum intPARTID supported is [MPAMF_PARTID_NRW_IDR, INTPARTID_MAX](#).

## Accessing the MPAMCFG_INTPARTID

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_INTPARTID_s must be accessible from the Secure MPAM feature page. MPAMCFG_INTPARTID_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_INTPARTID_s and MPAMCFG_INTPARTID_ns must be separate registers. The Secure instance (MPAMCFG_INTPARTID_s) accesses the PARTID narrowing used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_INTPARTID_ns) accesses the PARTID narrowing used for Non-secure PARTIDs.

**MPAMCFG_INTPARTID can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0600</td>
<td>MPAMCFG_INTPARTID_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0600</td>
<td>MPAMCFG_INTPARTID_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
**MPAMCFG_MBW_MAX, MPAM Memory Bandwidth Maximum Partition Configuration Register**

The MPAMCFG_MBW_MAX characteristics are:

**Purpose**

MPAMCFG_MBW_MAX is a 32-bit read-write register that controls the maximum fraction of memory bandwidth that the PARTID selected by `MPAMCFG_PART_SEL` is permitted to use. MPAMCFG_MBW_MAX controls maximum bandwidth for the Secure PARTID selected by the Secure instance of `MPAMCFG_PART_SEL`. MPAMCFG_MBW_MAX controls the maximum bandwidth for the Non-secure PARTID selected by the Non-secure instance of `MPAMCFG_PART_SEL`.

A PARTID that has used more than MAX is given no access to additional bandwidth if HARDLIM == 1 or is given additional bandwidth only if there are no requests from PARTIDs that have not exceeded their MAX if HARDLIM == 0.

**Configuration**

The power domain of MPAMCFG_MBW_MAX is **IMPLEMENTATION DEFINED**.

This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MAX == 1. Otherwise, direct accesses to MPAMCFG_MBW_MAX are **RES0**.

**Attributes**

MPAMCFG_MBW_MAX is a 32-bit register.

**Field descriptions**

The MPAMCFG_MBW_MAX bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| HARDLIM | RES0 | MAX |

**HARDLIM, bit [31]**

Hard bandwidth limiting.

<table>
<thead>
<tr>
<th>HARDLIM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When MAX bandwidth is exceeded, the partition contends with a low preference for downstream bandwidth beyond MAX.</td>
</tr>
<tr>
<td>0b1</td>
<td>When MAX bandwidth is exceeded, the partition does not be use any more bandwidth until the memory bandwidth measurement for the partition falls below MAX.</td>
</tr>
</tbody>
</table>

**Bits [30:16]**

Reserved, RES0.

**MAX, bits [15:0]**

Memory maximum bandwidth allocated to the partition selected by `MPAMCFG_PART_SEL`. MAX is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.
The implemented width of the fixed-point fraction is given in MPAMF_MBW_IDR.BWA_WD. Unimplemented bits are RAZ/WI. The implemented bits of the MAX field are always to the left of the field. For example, if BWA_WD = 3, the implemented bits are MPAMCFG_MBW_MAX[15:13] and MPAMCFG_MBW_MAX[12:0] are unimplemented.

The fixed-point fraction MAX is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is \(1 - \frac{1}{w}\).

**Accessing the MPAMCFG_MBW_MAX**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_MAX_s must be accessible from the Secure MPAM feature page. MPAMCFG_MBW_MAX_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_MAX_s and MPAMCFG_MBW_MAX_ns must be separate registers. The Secure instance (MPAMCFG_MBW_MAX_s) accesses the memory maximum bandwidth partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_MAX_ns) accesses the memory maximum bandwidth partitioning used for Non-secure PARTIDs.

**MPAMCFG_MBW_MAX can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0208</td>
<td>MPAMCFG_MBW_MAX_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0208</td>
<td>MPAMCFG_MBW_MAX_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MPAMCFG_MBW_MIN characteristics are:

**Purpose**

MPAMCFG_MBW_MIN is a 32-bit read-write register that controls the minimum fraction of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL is permitted to use. MPAMCFG_MBW_MIN_s controls the minimum bandwidth for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_MIN_ns controls the minimum bandwidth for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

A PARTID that has used less than MIN is given preferential access to bandwidth.

**Configuration**

The power domain of MPAMCFG_MBW_MIN is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MIN == 1. Otherwise, direct accesses to MPAMCFG_MBW_MIN are RES0.

**Attributes**

MPAMCFG_MBW_MIN is a 32-bit register.

**Field descriptions**

The MPAMCFG_MBW_MIN bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:16]**

Reserved, RES0.

**MIN, bits [15:0]**

Memory minimum bandwidth allocated to the partition selected by MPAMCFG_PART_SEL. MIN is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in MPAMF_MBW_IDR.BWA_WD. Unimplemented bits are RAZ/WI. The implemented bits of the MIN field are always to the left of the field. For example, if BWA_WD = 4, the implemented bits are MPAMCFG_MBW_MIN[15:12] and MPAMCFG_MBW_MIN[11:0] are unimplemented.

The fixed-point fraction MIN is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1 - 1/w.

**Accessing the MPAMCFG_MBW_MIN**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_MIN_s must be accessible from the Secure MPAM feature page. MPAMCFG_MBW_MIN_ns must be accessible from the Non-secure MPAM feature page.
MPAMCFG_MBW_MIN_s and MPAMCFG_MBW_MIN_ns must be separate registers. The Secure instance (MPAMCFG_MBW_MIN_s) accesses the memory minimum bandwidth partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_MIN_ns) accesses the memory minimum bandwidth partitioning used for Non-secure PARTIDs.

**MPAMCFG_MBW_MIN can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0200</td>
<td>MPAMCFG_MBW_MIN_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0200</td>
<td>MPAMCFG_MBW_MIN_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MPAMCFG_MBW_PBM characteristics are:

**Purpose**

The MPAMCFG_MBW_PBM register is a read-write register that configures the cache portions that a PARTID is allowed to allocate. MPAMCFG_MBW_PBM_s controls the bandwidth portion bitmap for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PBM_ns controls the bandwidth portion bitmap for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

After setting MPAMCFG_PART_SEL with a PARTID, software writes to the MPAMCFG_CPBM register to configure which cache portions the PARTID is allowed to allocate.

**Configuration**

The power domain of MPAMCFG_MBW_PBM is IMPLEMENTATIONDEFINED.

This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PBM == 1. Otherwise, direct accesses to MPAMCFG_MBW_PBM are RES0.

**Attributes**

MPAMCFG_MBW_PBM is a 4096-bit register.

**Field descriptions**

The MPAMCFG_MBW_PBM bit assignments are:

BWBM<n>, bit [n], for n = 0 to 4095

Each bit BWBM<n> grants permission to the PARTID to allocate bandwidth within bandwidth portion n.

<table>
<thead>
<tr>
<th>BWBM&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PARTID is not permitted to allocate into bandwidth portion n.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PARTID is permitted to allocate within bandwidth portion n.</td>
</tr>
</tbody>
</table>

The number of bits in the bandwidth portion partitioning bit map of this component is given in MPAMF_MBW_IDR.BWPBM_WD. BWPBM_WD contains a value from 1 to $2^{12}$, inclusive. Values of BWPBM_WD greater than 32 require a group of 32-bit registers to access the BWPBM, up to 128 32-bit registers.

Bits BWBM<n>, where n is greater than or equal to BWPBM_WD, are not required to be implemented.

**Accessing the MPAMCFG_MBW_PBM**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_PBM_s must be accessible from the Secure MPAM feature page. MPAMCFG_MBW_PBM_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_PBM_s and MPAMCFG_MBW_PBM_ns must be separate registers. The Secure instance (MPAMCFG_MBW_PBM_s) accesses the memory bandwidth portion bitmaps used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_PBM_ns) accesses the memory bandwidth portion bitmaps used for Non-secure PARTIDs.
**MPAMCFG_MBW_PBM** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x2000</td>
<td>MPAMCFG_MBW_PBM_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x2000</td>
<td>MPAMCFG_MBW_PBM_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MPAMCFG_MBW_PROP characteristics are:

**Purpose**

Controls the proportional stride of memory bandwidth that the PARTID selected by MPAMCFG_PART_SEL uses. MPAMCFG_MBW_PROP_s controls the bandwidth proportional stride for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_PROP_ns controls the bandwidth proportional stride for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

Proportional stride is a relative cost of bandwidth requested by one PARTID in relation to the costs of the bandwidths requested by each other PARTID also competing to use the bandwidth.

**Configuration**

The power domain of MPAMCFG_MBW_PROP is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PROP == 1. Otherwise, direct accesses to MPAMCFG_MBW_PROP are RES0.

**Attributes**

MPAMCFG_MBW_PROP is a 32-bit register.

**Field descriptions**

The MPAMCFG_MBW_PROP bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>EN</td>
<td>Enable proportional stride bandwidth partitioning.</td>
</tr>
<tr>
<td>30</td>
<td>RES0</td>
<td>Reserved</td>
</tr>
<tr>
<td>29</td>
<td>STRIDEM1</td>
<td>Memory bandwidth stride minus 1 allocated to the partition selected by MPAMCFG_PART_SEL. STRIDEM1 represents the normalized cost of bandwidth consumption by the partition.</td>
</tr>
</tbody>
</table>

Enable proportional stride bandwidth partitioning.

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The selected partition is not regulated by proportional stride bandwidth partitioning.</td>
</tr>
<tr>
<td>0b1</td>
<td>The selected partition has bandwidth usage regulated by proportional stride bandwidth partitioning as controlled by STRIDEM1.</td>
</tr>
</tbody>
</table>

Reserved, RES0.

**STRIDEM1, bits [15:0]**

Memory bandwidth stride minus 1 allocated to the partition selected by MPAMCFG_PART_SEL. STRIDEM1 represents the normalized cost of bandwidth consumption by the partition.

The proportional stride partitioning control parameter is an unsigned integer representing the normalized cost to a partition for consuming bandwidth. Larger values have a larger cost and correspond to a lesser allocation of bandwidth while smaller values indicate a lesser cost and therefore a higher allocation of bandwidth.
The implemented width of STRIDEM1 is given in MPAMF_MBW_IDR_BWA_WD.

**Accessing the MPAMCFG_MBW_PROP**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_PROP_s must be accessible from the Secure MPAM feature page. MPAMCFG_MBW_PROP_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_PROP_s and MPAMCFG_MBW_PROP_ns must be separate registers. The Secure instance (MPAMCFG_MBW_PROP_s) accesses the memory proportional stride bandwidth partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_PROP_ns) accesses the memory proportional stride bandwidth partitioning used for Non-secure PARTIDs.

**MPAMCFG_MBW_PROP can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0500</td>
<td>MPAMCFG_MBW_PROP_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0500</td>
<td>MPAMCFG_MBW_PROP_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
MPAMCFG_MBW_WINWD, MPAM Memory Bandwidth Partitioning Window Width Configuration Register

The MPAMCFG_MBW_WINWD characteristics are:

**Purpose**

MPAMCFG_MBW_WINWD is a 32-bit register that shows and sets the value of the window width for the PARTID in MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_s reads and controls the bandwidth control window width for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_MBW_WINWD_ns reads and controls the bandwidth control window for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

MPAMCFG_MBW_WINWD is read-only if MPAMF_MBW_IDR.WINDWR == 0, and the window width is set by the hardware, even if variable.

MPAMCFG_MBW_WINWD is read-write if MPAMF_MBW_IDR.WINDWR == 1, permitting configuration of the window width for each PARTID independently on hardware that supports this functionality.

**Configuration**

The power domain of MPAMCFG_MBW_WINWD is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMCFG_MBW_WINWD are RES0.

**Attributes**

MPAMCFG_MBW_WINWD is a 32-bit register.

**Field descriptions**

The MPAMCFG_MBW_WINWD bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:24</td>
<td>Reserved, RES0</td>
<td></td>
</tr>
<tr>
<td>23:8</td>
<td>Window width, integer microseconds.</td>
<td>US_INT</td>
</tr>
<tr>
<td>7:0</td>
<td>Window width, fractional microseconds.</td>
<td>US_FRAC</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**US_INT, bits [23:8]**

Window width, integer microseconds.

This field reads (and sets) the integer part of the window width in microseconds for the PARTID selected by MPAMCFG_PART_SEL.

**US_FRAC, bits [7:0]**

Window width, fractional microseconds.

This field reads (and sets) the fractional part of the window width in microseconds for the PARTID selected by MPAMCFG_PART_SEL.
Accessing the MPAMCFG_MBW_WINWD

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_MBW_WINWD_s must be accessible from the Secure MPAM feature page. MPAMCFG_MBW_WINWD_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_MBW_WINWD_s and MPAMCFG_MBW_WINWD_ns must be separate registers. The Secure instance (MPAMCFG_MBW_WINWD_s) accesses the window width used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_MBW_WINWD_ns) accesses the window width used for Non-secure PARTIDs.

**MPAMCFG_MBW_WINWD can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0220</td>
<td>MPAMCFG_MBW_WINWD_s</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are RO.
- Otherwise accesses to this register are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0220</td>
<td>MPAMCFG_MBW_WINWD_ns</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are RO.
- Otherwise accesses to this register are RW.
MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register

The MPAMCFG_PART_SEL characteristics are:

**Purpose**

Selects a partition ID to configure. MPAMCFG_PART_SEL_s selects a Secure PARTID to configure. MPAMCFG_PART_SEL_ns selects a Non-secure PARTID to configure.

After setting this register with a PARTID, software (usually a hypervisor) can perform a series of accesses to MPAMCFG registers to configure parameters for MPAM resource controls to use when requests have that PARTID.

**Configuration**

The power domain of MPAMCFG_PART_SEL is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_CCAP_PART == 1, or MPAMF_IDR.HAS_CPOR_PART == 1, or MPAMF_IDR.HAS_MBW_PART == 1, or MPAMF_IDR.HAS_PRI_PART == 1, or MPAMF_IDR.HAS_PARTID_NRW == 1, or (MPAMF_IDR.EXT == 0 and MPAMF_IDR.HAS_IMPL_IDR == 1) or (MPAMF_IDR.EXT == 1, MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.NO_IMPL_PART == 0). Otherwise, direct accesses to MPAMCFG_PART_SEL are RES0.

**Attributes**

MPAMCFG_PART_SEL is a 32-bit register.

**Field descriptions**

The MPAMCFG_PART_SEL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RIS | RES0 | INTERNAL | PARTID_SEL |

**Bits [31:28]**

Reserved, RES0.

**RIS, bits [27:24]**

When ARMv8.6-MPAM is implemented, MPAMF_IDR.EXT == 1 and MAPMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MPAMCFG registers and describe with MPAMF ID registers.

Otherwise:

Reserved, RES0.

**Bits [23:17]**

Reserved, RES0.
INTERNAL, bit [16]

Internal PARTID.

If MPAMF_IDR.HAS_PARTID_NRW = 0, this field is RAZ/WI.

If MPAMF_IDR.HAS_PARTID_NRW = 1:

<table>
<thead>
<tr>
<th>INTERNAL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PARTID_SEL is interpreted as a request PARTID and ignored except for use with MPAMCFG_INTPARTID register access.</td>
</tr>
<tr>
<td>0b1</td>
<td>PARTID_SEL is interpreted as an internal PARTID and used for access to MPAMCFG control settings except for MPAMCFG_INTPARTID.</td>
</tr>
</tbody>
</table>

If PARTID narrowing is implemented as indicated by MPAMF_IDR.HAS_PARTID_NRW = 1, when accessing other MPAMCFG registers the value of the MPAMCFG_PART_SEL.INTERNAL bit is checked for these conditions:

- When the MPAMCFG_INTPARTID register is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 0, an Unexpected_INTERNAL error is set in MPAMF_ESR.
- When an MPAMCFG register other than MPAMCFG_INTPARTID is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 1, MPAMF_ESR is set to indicate an intPARTID_Range error.

In either error case listed here, the value returned by a read operation is UNPREDICTABLE, and the control settings are not affected by a write.

PARTID_SEL, bits [15:0]

Selects the partition ID to configure.

Reads and writes to other MPAMCFG registers are indexed by PARTID_SEL and by the NS bit used to access MPAMCFG_PART_SEL to access the configuration for a single partition.

Accessing the MPAMCFG_PART_SEL

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_PART_SEL_s must be accessible from the Secure MPAM feature page. MPAMCFG_PART_SEL_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_PART_SEL_s and MPAMCFG PART_SEL_ns must be separate registers. The Secure instance (MPAMCFG_PART_SEL_s) accesses the PARTID selector used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_PART_SEL_ns) accesses the PARTID selector used for Non-secure PARTIDs.

MPAMCFG_PART_SEL can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0100</td>
<td>MPAMCFG_PART_SEL_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0100</td>
<td>MPAMCFG_PART_SEL_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
MPAMCFG_PRI, MPAM Priority Partition Configuration Register

The MPAMCFG_PRI characteristics are:

**Purpose**

MPAMCFG_PRI is a 32-bit read-write register that controls the internal and downstream priority of requests attributed to the PARTID selected by MPAMCFG_PART_SEL. MPAMCFG_PRI controls the priorities for the Secure PARTID selected by the Secure instance of MPAMCFG_PART_SEL. MPAMCFG_PRI_ns controls the priorities for the Non-secure PARTID selected by the Non-secure instance of MPAMCFG_PART_SEL.

**Configuration**

The power domain of MPAMCFG_PRI is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMCFG_PRI are RES0.

**Attributes**

MPAMCFG_PRI is a 32-bit register.

**Field descriptions**

The MPAMCFG_PRI bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>DSPRI</td>
</tr>
<tr>
<td>15</td>
<td>INTPRI</td>
</tr>
</tbody>
</table>

**DSPRI, bits [31:16]**

Downstream priority.

If MPAMF_PRI_IDR.HAS_DSPRI == 0, bits of this field are RES0 as this field is not used.

If MPAMF_PRI_IDR.HAS_DSPRI == 1, this field is a priority value applied to downstream communications from this MSC for transactions of the partition selected by MPAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.DSPRI_WD bits.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of MPAMF_PRI_IDR.DSPRI_0_IS_LOW.

**INTPRI, bits [15:0]**

Internal priority.

If MPAMF_PRI_IDR.HAS_INTPRI == 0, bits of this field are RES0 as this field is not used.

If MPAMF_PRI_IDR.HAS_INTPRI == 1, this field is a priority value applied internally inside this MSC for transactions of the partition selected by Mext-PAMCFG_PART_SEL.

The implemented width of this field is MPAMF_PRI_IDR.INTPRI_WD bits.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of MPAMF_PRI_IDR.INTPRI_0_IS_LOW.
Accessing the MPAMCFG_PRI

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_PRI_s must be accessible from the Secure MPAM feature page. MPAMCFG_PRI_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_PRI_s and MPAMCFG_PRI_ns must be separate registers. The Secure instance (MPAMCFG_PRI_s) accesses the priority partitioning used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_PRI_ns) accesses the priority partitioning used for Non-secure PARTIDs.

MPAMCFG_PRI can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0400</td>
<td>MPAMCFG_PRI_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0400</td>
<td>MPAMCFG_PRI_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

27/09/2019 18:23, 6134483bd14d8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MPAMF_AIDR, MPAM Architecture Identification Register

The MPAMF_AIDR characteristics are:

**Purpose**

Identifies the version of the MPAM architecture that this MSC implements.

Note: The following values are defined for bits [7:0]:

- 0x01 == MPAM architecture v0.1
- 0x10 == MPAM architecture v1.0
- 0x11 == MPAM architecture v1.1

**Configuration**

The power domain of MPAMF_AIDR is **IMPLEMENTATION DEFINED**.

**Attributes**

MPAMF_AIDR is a 32-bit register.

**Field descriptions**

The MPAMF_AIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>ArchMajorRev</td>
</tr>
<tr>
<td>29</td>
<td>ArchMinorRev</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**ArchMajorRev, bits [7:4]**

Major revision of the MPAM architecture implemented by the MSC.

This table shows the only valid combinations of MPAM version numbers in an MSC. FORCE_NS functionality is only available in MPAM v0.1.

<table>
<thead>
<tr>
<th>ArchMajorRev</th>
<th>ArchMinorRev</th>
<th>MPAMv</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>v0.1</td>
<td>None.</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>v0.1</td>
<td>MPAMv1.0 + MPAMv1.1 + FORCE_NS</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>v1.0</td>
<td>MPAMv1.0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>v1.1</td>
<td>MPAMv1.0 + MPAMv1.1 - FORCE_NS</td>
</tr>
</tbody>
</table>

**ArchMinorRev, bits [3:0]**

Minor revision of the MPAM architecture implemented by the MSC.

See the table in the description of the ArchMajorRev field in this register.
Accessing the MPAMF_AIDR

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_AIDR is read-only.

MPAMF_AIDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_AIDR must have the same contents in the Secure and Non-secure MPAM feature pages.

**MPAMF_AIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0020</td>
<td>MPAMF_AIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0020</td>
<td>MPAMF_AIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
MPAMF CCAP IDR, MPAM Features Cache Capacity Partitioning ID register

The MPAMF CCAP IDR characteristics are:

**Purpose**

The MPAMF CCAP IDR is a 32-bit read-only register that indicates the number of fractional bits in MPAMCFG CMAX for this MSC. MPAMF CCAP IDR s indicates the number of fractional bits in the Secure instance of MPAMCFG CMAX. MPAMF CCAP IDR ns indicates the number of fractional bits in the Non-secure instance of MPAMCFG CMAX.

**Configuration**

The power domain of MPAMF CCAP IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF IDR HAS CCAP PART == 1. Otherwise, direct accesses to MPAMF CCAP IDR are RES0.

**Attributes**

MPAMF CCAP IDR is a 32-bit register.

**Field descriptions**

The MPAMF CCAP IDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMAX WD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:6]**

Reserved, RES0.

**CMAX WD, bits [5:0]**

Number of fractional bits implemented in the cache capacity partitioning control, MPAMCFG CMAX, of this device. See MPAMCFG CMAX.

This field must contain a value from 1 to 16, inclusive.

**Accessing the MPAMF CCAP IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF CCAP IDR is read-only.

MPAMF CCAP IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF CCAP IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF CCAP IDR s) and Non-secure (MPAMF CCAP IDR ns) MPAM feature pages.
MPAMF_CCAP_IDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0038</td>
<td>MPAMF_CCAP_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0038</td>
<td>MPAMF_CCAP_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The MPAMF_CPOR_IDR characteristics are:

**Purpose**

The MPAMF_CPOR_IDR is a 32-bit read-only register that indicates the number of bits in MPAMCFG_CPBM for this MSC. MPAMF_CPOR_IDR_s indicates the number of bits in the Secure instance of MPAMCFG_CPBM. MPAMF_CPOR_IDR_ns indicates the number of bits in the Non-secure instance of MPAMCFG_CPBM.

**Configuration**

The power domain of MPAMF_CPOR_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_CPOR == 1. Otherwise, direct accesses to MPAMF_CPOR_IDR are RES0.

**Attributes**

MPAMF_CPOR_IDR is a 32-bit register.

**Field descriptions**

The MPAMF_CPOR_IDR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>CPBM_WD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**CPBM_WD, bits [15:0]**

Number of bits in the cache portion partitioning bit map of this device. See MPAMCFG_CPBM.

This field must contain a value from 1 to 32768, inclusive. Values greater than 32 require a group of 32-bit registers to access the CPBM, up to 1024 if CPBM_WD is the largest value.

**Accessing the MPAMF_CPOR_IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_CPOR_IDR is read-only.

MPAMF_CPOR_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_CPOR_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_CPOR_IDR_s) and Non-secure (MPAMF_CPOR_IDR_ns) MPAM feature pages.
MPAMF_CPOR_IDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0030</td>
<td>MPAMF_CPOR_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0030</td>
<td>MPAMF_CPOR_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
### Purpose

The MPAMF_CSUMON_IDR is a 32-bit read-only register that indicates the number of cache storage usage monitors for this MSC. MPAMF_CSUMON_IDR_s indicates the number of Secure cache storage usage monitors. MPAMF_CSUMON_IDR_ns indicates the number of Non-secure cache storage usage monitors.

### Configuration

The power domain of MPAMF_CSUMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MPAMF_CSUMON_IDR are RES0.

### Attributes

MPAMF_CSUMON_IDR is a 32-bit register.

### Field descriptions

The MPAMF_CSUMON_IDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| HAS_CAPTURE | RES0 | NUM_MON |

**HAS_CAPTURE, bit [31]**

The implementation of this MSC supports copying an MSMON_CSU to the corresponding MSMON_CSU_CAPTURE on a capture event.

<table>
<thead>
<tr>
<th>HAS_CAPTURE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSMON_CSU_CAPTURE is not implemented and there is no support for capture events in the CSU monitor.</td>
</tr>
<tr>
<td>0b1</td>
<td>The MSMON_CSU_CAPTURE register is implemented and the CSU monitor supports the capture event behavior.</td>
</tr>
</tbody>
</table>

**Bits [30:16]**

Reserved, RES0.

**NUM_MON, bits [15:0]**

The number of cache storage usage monitors implemented in this MSC.

### Accessing the MPAMF_CSUMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_CSUMON_IDR is read-only.
MPAMF-CSUMON_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF-CSUMON_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF-CSUMON_IDR_s) and Non-secure (MPAMF-CSUMON_IDR_ns) MPAM feature pages.

**MPAMF-CSUMON_IDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0088</td>
<td>MPAMF-CSUMON_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0088</td>
<td>MPAMF-CSUMON_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

---

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MPAMF_ECR, MPAM Error Control Register

The MPAMF_ECR characteristics are:

**Purpose**

MPAMF_ECR is a 32-bit read-write register that controls MPAM error interrupts for this MSC. MPAMF_ECR_s controls Secure MPAM error handling. MPAMF_ECR_ns controls Non-secure MPAM error handling.

**Configuration**

The power domain of MPAMF_ECR is IMPLEMENTATION DEFINED.

If a MSC cannot encounter any of the error conditions listed in section 15.1, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

**Attributes**

MPAMF_ECR is a 32-bit register.

**Field descriptions**

The MPAMF_ECR bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

- **Reserved, RES0**
- **INTEN**, bit [0]

**INTEN, bit [0]**

Interrupt Enable.

<table>
<thead>
<tr>
<th>INTEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAM error interrupts are not generated.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAM error interrupts are generated.</td>
</tr>
</tbody>
</table>

**Accessing the MPAMF_ECR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_ECR_s must be accessible from the Secure MPAM feature page. MPAMF_ECR_ns must be accessible from the Non-secure MPAM feature page.

MPAMF_ECR_s and MPAMF_ECR_ns must be separate registers. The Secure instance (MPAMF_ECR_s) accesses the error interrupt controls used for Secure PARTIDs, and the Non-secure instance (MPAMF_ECR_ns) accesses the error interrupt controls used for Non-secure PARTIDs.

**MPAMF_ECR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x00F0</td>
<td>MPAMF_ECR_s</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0xFF0</td>
<td>MPAMF_ECR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
MPAMF_ESR, MPAM Error Status Register

The MPAMF_ESR characteristics are:

**Purpose**

Indicates MPAM error status for this MSC. MPAMF_ESR_s reports Secure MPAM errors. MPAMF_ESR_ns reports Non-secure MPAM errors.

Software should write this register after reading the status of an error to reset ERRCODE to 0x0000 and OVRWR to 0 so that future errors are not reported with OVRWR set.

**Configuration**

The power domain of MPAMF_ESR is IMPLEMENTATION DEFINED.

If a MSC cannot encounter any of the error conditions listed in Chapter 12 Errors in MSCs, both the MPAMF_ESR and MPAMF_ECR must be RAZ/WI.

**Attributes**

MAMPF_ESR is 64-bit register when ARMv8.6-MPAM is implemented and MPAMF_IDR.HAS_EXTD_ESR == 1.

Otherwise, MAMPF_ESR is a 32-bit register.

**Field descriptions**

The MPAMF_ESR bit assignments are:

When ARMv8.6-MPAM is implemented and MPAMF_IDR.HAS_EXTD_ESR == 1:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| OVRWR | RES0 | ERRCODE | PMG | PARTID_MON |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

**Bits [63:36]**

Reserved, RES0.

**RIS, bits [35:32]**

When MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. Where applicable to the ERRCODE, captures the RIS value for the error.

Otherwise:

Reserved, RES0.

**OVRWR, bit [31]**

Overwritten.

If 0 and ERRCODE == 0b0000, no errors have occurred.
If 0 and ERRCODE is non-zero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is non-zero, multiple errors have occurred and this register records the most recent error.

The state where this bit is 1 and ERRCODE is zero must not be produced by hardware and is only reached when software writes this combination into this register.

**Bits [30:28]**

Reserved, RES0.

**ERRCODE, bits [27:24]**

Error code.

<table>
<thead>
<tr>
<th>ERRCODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No error.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PARTID_SEL Range.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Req_PARTID Range.</td>
</tr>
<tr>
<td>0b0011</td>
<td>MSMONCFG_ID_RANGE.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Req_PMG Range.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Monitor Range.</td>
</tr>
<tr>
<td>0b0110</td>
<td>intPARTID Range.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Unexpected INTERNAL.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Undefined_RIS_PART_SEL.</td>
</tr>
<tr>
<td>0b1001</td>
<td>RIS_No_Control.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Undefined_RIS_MON_SEL.</td>
</tr>
<tr>
<td>0b1011</td>
<td>RIS_No_Monitor.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**PMG, bits [23:16]**

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to 0x00 on an error that does not capture PMG.

**PARTID_MON, bits [15:0]**

PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to 0.

**Otherwise:**

|    | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| OVRWR | RES0 | ERRCODE | PMG | PARTID_MON |

**OVRWR, bit [31]**

Overwritten.

If 0 and ERRCODE == 0b0000, no errors have occurred.

If 0 and ERRCODE is non-zero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is non-zero, multiple errors have occurred and this register records the most recent error.
The state where this bit is 1 and ERRCODE is 0 must not be produced by hardware and is only reached when software writes this combination into this register.

**Bits [30:28]**

Reserved, RES0.

**ERRCODE, bits [27:24]**

Error code.

<table>
<thead>
<tr>
<th>ERRCODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No error.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PARTID_SEL_Range.</td>
</tr>
<tr>
<td>0b0010</td>
<td>Req_PARTID_Range.</td>
</tr>
<tr>
<td>0b0011</td>
<td>MSMONCFG_ID_RANGE.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Req_PMG_Range.</td>
</tr>
<tr>
<td>0b0101</td>
<td>Monitor_Range.</td>
</tr>
<tr>
<td>0b0110</td>
<td>intPARTID_Range.</td>
</tr>
<tr>
<td>0b0111</td>
<td>Unexpected_INTERNAL.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1001</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1010</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1011</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1100</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1101</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1110</td>
<td>Reserved.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

**PMG, bits [23:16]**

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to 0x00 on an error that does not capture PMG.

**PARTID_MON, bits [15:0]**

PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to 0x0000.

**Accessing the MPAMF_ESR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_ESR_s must be accessible from the Secure MPAM feature page. MPAMF_ESR_ns must be accessible from the Non-secure MPAM feature page.

MPAMF_ESR_s and MPAMF_ESR_ns must be separate registers. The Secure instance (MPAMF_ESR_s) accesses the error status used for Secure PARTIDs, and the Non-secure instance (MPAMF_ESR_ns) accesses the error status used for Non-secure PARTIDs.

**MPAMF_ESR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x00F8</td>
<td>MPAMF_ESR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x00F8</td>
<td>MPAMF_ESR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
MPAMF_IDR, MPAM Features Identification Register

The MPAMF_IDR characteristics are:

Purpose

Indicates which memory partitioning and monitoring features are present on this MSC. MPAMF_IDR_s indicates the MPAM features accessed from the Secure MPAM feature page. MPAMF_IDR_ns indicates the MPAM features accessed from the Non-secure MPAM feature page.

Configuration

The power domain of MPAMF_IDR is IMPLEMENTATION DEFINED.

Attributes

MAMPF_IDR is 64-bit register when ARMv8.6-MPAM is implemented.

Otherwise, MAMPF_IDR is a 32-bit register.

Field descriptions

The MPAMF_IDR bit assignments are:

When ARMv8.6-MPAM is implemented:

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>61</th>
<th>60</th>
<th>59</th>
<th>58</th>
<th>57</th>
<th>56</th>
<th>55</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>HAS_PARTITION_NR</td>
<td>HAS_MSMON</td>
<td>HAS_IMPL_IDR</td>
<td>EXT</td>
<td>HAS_PRI_PART</td>
<td>HAS_MBW_PART</td>
<td>HAS_CPOR_PART</td>
<td>HAS_CCAP_PART</td>
</tr>
</tbody>
</table>

Bits [63:60]

Reserved, RES0.

RIS_MAX, bits [59:56]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Maximum RIS value supported in MPAMCFG_PART_SEL. Must be 0b0000 if MPAMF_IDR.HAS_RIS == 0.

Otherwise:

Reserved, RES0.

Bits [55:40]

Reserved, RES0.

HAS_ESR, bit [39]

When MPAMF_IDR.EXT == 1:

MPAMF_ESR is implemented.
<table>
<thead>
<tr>
<th>HAS_ESR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMF_ESR, MPAM_ECR and MPAM error handling are not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMF_ESR, MPAM_ECR and MPAM error handling are implemented.</td>
</tr>
</tbody>
</table>

Otherwise:
Reserved, RES0.

**HAS_EXTD_ESR, bit [38]**

When MPAMF_IDR.EXT == 1:

MPAMF_ESR is 64 bits.

<table>
<thead>
<tr>
<th>HAS_EXTD_ESR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMF_ESR is 32 bits.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMF_ESR is 64 bits.</td>
</tr>
</tbody>
</table>

Otherwise:
Reserved, RES0.

**NO_IMPL_MSMON, bit [37]**

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

MPAMF_IMPL_IDR defines no IMPLEMENTATION DEFINED resource monitors.

<table>
<thead>
<tr>
<th>NO_IMPL_MSMON</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource monitor.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource monitors.</td>
</tr>
</tbody>
</table>

Otherwise:
Reserved, RES0.

**NO_IMPL_PART, bit [36]**

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

MPAMF_IMPL_IDR defines no IMPLEMENTATION DEFINED resource controls.

<table>
<thead>
<tr>
<th>NO_IMPL_PART</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource control.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource controls.</td>
</tr>
</tbody>
</table>

Otherwise:
Reserved, RES0.

**Bits [35:33]**

Reserved, RES0.
HAS_RIS, bit [32]

When MPAMF_IDR.EXT == 1:

Has resource instance selector. Indicates that MPAMCFG_PART_SEL contains the RIS field that selects a resource instance to control.

<table>
<thead>
<tr>
<th>HAS_RIS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMCFG_PART_SEL does not implement the MPAMFCFG_PART_SEL.RIS field or multiple resource instance support.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMCFG_PART_SEL implements the MPAMFCFG_PART_SEL.RIS field and MPAM resource instance numbers up to MPAMF_IDR.RIS_MAX.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

HAS_PARTID_NRW, bit [31]

Has PARTID narrowing.

<table>
<thead>
<tr>
<th>HAS_PARTID_NRW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not have MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID or intPARTID mapping support.</td>
</tr>
<tr>
<td>0b1</td>
<td>Supports the MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID registers.</td>
</tr>
</tbody>
</table>

HAS_MSMON, bit [30]

Has resource monitors. Indicates whether this MSC has MPAM resource monitors.

<table>
<thead>
<tr>
<th>HAS_MSMON</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR.</td>
</tr>
<tr>
<td>0b1</td>
<td>Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR.</td>
</tr>
</tbody>
</table>

HAS_IMPL_IDR, bit [29]

Has MPAMF_IMPL_IDR. Indicates whether this MSC has the implementation-specific MPAM features register, MPAMF_IMPL_IDR.

<table>
<thead>
<tr>
<th>HAS_IMPL_IDR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not have MPAMF_IMPL_IDR.</td>
</tr>
<tr>
<td>0b1</td>
<td>Has MPAMF_IMPL_IDR.</td>
</tr>
</tbody>
</table>

EXT, bit [28]

Extended MPAMF_IDR.

<table>
<thead>
<tr>
<th>EXT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMF_IDR has bits defined in [63:32]. The register is 64-bits.</td>
</tr>
</tbody>
</table>

HAS_PRI_PART, bit [27]

Has priority partitioning. Indicates whether this MSC implements MPAM priority partitioning and MPAMF_PRI_IDR.
**HAS_PRI_PART**

- **[0b0]** Does not support priority partitioning or have MPAMF_PRI_IDR.
- **[0b1]** Has MPAMF_PRI_IDR.

**HAS_MBW_PART, bit [26]**

Has memory bandwidth partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and MPAMF_MBW_IDR.

- **[0b0]** Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register.
- **[0b1]** Has MPAMF_MBW_IDR register.

**HAS_CPOR_PART, bit [25]**

Has cache portion partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and MPAMF_CPOR_IDR.

- **[0b0]** Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPBM registers.
- **[0b1]** Has MPAMF_CPOR_IDR and MPAMCFG_CPBM registers.

**HAS_CCAP_PART, bit [24]**

Has cache capacity partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

- **[0b0]** Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.
- **[0b1]** Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers.

**PMG_MAX, bits [23:16]**

Maximum value of Non-secure PMG supported by this component.

**PARTID_MAX, bits [15:0]**

Maximum value of Non-secure PARTID supported by this component.

**Otherwise:**

**HAS_PARTID_NRW, bit [31]**

Has PARTID narrowing.

- **[0b0]** Does not have MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID or intPARTID mapping support.
- **[0b1]** Supports the MPAMF_PARTID_NRW_IDR, MPAMCFG_INTPARTID registers.
HAS_MSMON, bit [30]

Has resource monitors. Indicates whether this MSC has MPAM resource monitors.

<table>
<thead>
<tr>
<th>HAS_MSMON</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR.</td>
</tr>
<tr>
<td>0b1</td>
<td>Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR.</td>
</tr>
</tbody>
</table>

HAS_IMPL_IDR, bit [29]

Has MPAMF_IMPL_IDR. Indicates whether this MSC has the implementation-specific MPAM features register, MPAMF_IMPL_IDR.

<table>
<thead>
<tr>
<th>HAS_IMPL_IDR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not have MPAMF_IMPL_IDR.</td>
</tr>
<tr>
<td>0b1</td>
<td>Has MPAMF_IMPL_IDR.</td>
</tr>
</tbody>
</table>

EXT, bit [28]

From Armv8.6:

Extended MPAMF_IDR.

<table>
<thead>
<tr>
<th>EXT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits.</td>
</tr>
<tr>
<td>0b1</td>
<td>MPAMF_IDR has bits defined in [63:32]. The register is 64-bits.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has priority partitioning. Indicates whether this MSC implements MPAM priority partitioning and MPAMF_PRI_IDR.

<table>
<thead>
<tr>
<th>HAS_PRI_PART</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not support priority partitioning or have MPAMF_PRI_IDR.</td>
</tr>
<tr>
<td>0b1</td>
<td>Has MPAMF_PRI_IDR.</td>
</tr>
</tbody>
</table>

HAS_MBW_PART, bit [26]

Has memory bandwidth partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and MPAMF_MBW_IDR.

<table>
<thead>
<tr>
<th>HAS_MBW_PART</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register.</td>
</tr>
<tr>
<td>0b1</td>
<td>Has MPAMF_MBW_IDR register.</td>
</tr>
</tbody>
</table>

HAS_CPOR_PART, bit [25]

Has cache portion partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and MPAMF_CPOR_IDR.
**HAS_CPOR_PART**

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

**HAS_CCAP_PART**, bit [24]

Has cache capacity partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the MPAM_CCAP_IDR and MPAMCFG_CMAX registers.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

**PMG_MAX**, bits [23:16]

Maximum value of Non-secure PMG supported by this component.

**PARTID_MAX**, bits [15:0]

Maximum value of Non-secure PARTID supported by this component.

**Accessing the MPAMF_IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_IDR is read-only.

MPAMF_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_IDR_s) and Non-secure (MPAMF_IDR_ns) MPAM feature pages.

**MPAMF_IDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0000</td>
<td>MPAMF_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0000</td>
<td>MPAMF_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The MPAM_IIDR characteristics are:

**Purpose**

Uniquely identifies the MSC implementation by the combination of implementer, product ID, variant and revision.

**Configuration**

The power domain of MPAM_IIDR is **IMPLEMENTATION DEFINED**.

**Attributes**

MPAM_IIDR is a 32-bit register.

**Field descriptions**

The MPAM_IIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Bit Assignments</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ProductID</td>
<td>IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>20</td>
<td>Variant</td>
<td>IMPLEMENTATION DEFINED value used to distinguish product variants, or major revisions of the product.</td>
</tr>
<tr>
<td>11</td>
<td>Revision</td>
<td>IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product.</td>
</tr>
</tbody>
</table>

**Note**

Implementations of ProductID with differing software interfaces are expected to have different values in the MPAM_IIDR.Variant field.

**Revision, bits [15:12]**

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED value used to distinguish minor revisions of the product.

**Note**

This field is intended to differentiate product revisions that are minor changes and are largely software compatible with previous revisions.
MPAMF_IIDR, MPAM Implementation Identification Register

**Implementer, bits [11:0]**

Contains the JEP106 code of the company that implemented the MPAM MSC.

[11:8] must contain the JEP106 continuation code of the implementer.

[7] must always be 0.

[6:0] must contain the JEP106 identity code of the implementer.

For an Arm implementation, bits[11:0] are 0x43B.

**Accessing the MPAMF_IIDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_IIDR is read-only.

MPAMF_IIDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_IIDR must have the same contents in the Secure and Non-secure MPAM feature pages.

**MPAMF_IIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0018</td>
<td>MPAMF_IIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0018</td>
<td>MPAMF_IIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
**MPAMF_IMPL_IDR, MPAM Implementation-Specific Partitioning Feature Identification Register**

The MPAMF_IMPL_IDR characteristics are:

**Purpose**

The MPAMF_IMPL_IDR is a 32-bit read-only register that indicates the implementation-defined partitioning and monitoring features and parameters of the MSC. MPAMF_IMPL_IDR_s indicates implementation-defined partitioning and monitoring features accessed from the Secure MPAM feature page. MPAMF_IMPL_IDR_ns indicates those accessed from the Non-secure MPAM feature page.

**Configuration**

The power domain of MPAMF_IMPL_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MPAMF_IMPL_IDR are RES0.

**Attributes**

MPAMF_IMPL_IDR is a 32-bit register.

**Field descriptions**

The MPAMF_IMPL_IDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | IMP DEF |

**IMP DEF, bits [31:0]**

IMPLEMENTATION DEFINED.

All 32 bits of this register are available to be used as the implementer sees fit to indicate the presence of implementation-defined MPAM features in this MSC and to give additional implementation-specific read-only information about the parameters of implementation-specific MPAM features to software.

**Accessing the MPAMF_IMPL_IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_IMPL_IDR is read-only.

MPAMF_IMPL_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_IMPL_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_IMPL_IDR_s) and Non-secure (MPAMF_IMPL_IDR_ns) MPAM feature pages.

**MPAMF_IMPL_IDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0028</td>
<td>MPAMF_IMPL_IDR_s</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0028</td>
<td>MPAMF_IMPL_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
MPAMF_MBW_IDR, MPAM Memory Bandwidth Partitioning Identification Register

The MPAMF_MBW_IDR characteristics are:

**Purpose**

The MPAMF_MBW_IDR is a 32-bit read-only register that indicates which MPAM bandwidth partitioning features are present on this MSC. MPAMF_MBW_IDR_s indicates bandwidth partitioning features accessed from the Secure MPAM feature page. MPAMF_MBW_IDR_ns indicates bandwidth partitioning features accessed from the Non-secure MPAM feature page.

**Configuration**

The power domain of MPAMF_MBW_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMF_MBW_IDR are RES0.

**Attributes**

MPAMF_MBW_IDR is a 32-bit register.

**Field descriptions**

The MPAMF_MBW_IDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-29</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>28-16</td>
<td>Bandwidth portion bitmap width.</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>14</td>
<td>Indicates the bandwidth accounting period register is writable.</td>
</tr>
</tbody>
</table>

**Bits [31:29]**

Reserved, RES0.

**BWPBM_WD, bits [28:16]**

Bandwidth portion bitmap width.

The number of bandwidth portion bits in MPAMCFG_MBW_PBM.BWPBM.

This field must contain a value from 1 to 4096, inclusive. Values greater than 32 require a group of 32-bit registers to access the BWPBM, up to 128 if BWPBM_WD is the largest value.

**Bit [15]**

Reserved, RES0.

**WINDWR, bit [14]**

Indicates the bandwidth accounting period register is writable.
### WINDWR

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The bandwidth accounting period is readable from MPAMCFG_MBW_WINWD which might be fixed or vary due to clock rate reconfiguration of the memory channel or memory controller.</td>
</tr>
<tr>
<td>0b1</td>
<td>The bandwidth accounting width is readable and writable per partition in MPAMCFG_MBW_WINWD.</td>
</tr>
</tbody>
</table>

### HAS_PROP, bit [13]

Indicates that this MSC implements proportional stride bandwidth partitioning and the MPAMCFG_MBW_PROP register.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no memory bandwidth proportional stride control and no MPAMCFG_MBW_PROP register.</td>
</tr>
<tr>
<td>0b1</td>
<td>The MPAMCFG_MBW_PROP register exists and the proportional stride memory bandwidth allocation scheme is supported.</td>
</tr>
</tbody>
</table>

### HAS_PBM, bit [12]

Indicates that this MSC implements bandwidth portion partitioning and the MPAMCFG_MBW_PBM register.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no memory bandwidth portion control and no MPAMCFG_MBW_PBM register.</td>
</tr>
<tr>
<td>0b1</td>
<td>The MPAMCFG_MBW_PBM register exists and the memory bandwidth portion allocation scheme exists.</td>
</tr>
</tbody>
</table>

### HAS_MAX, bit [11]

Indicates that this MSC implements maximum bandwidth partitioning and the MPAMCFG_MBW_MAX register.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no maximum memory bandwidth control and no MPAMCFG_MBW_MAX register.</td>
</tr>
<tr>
<td>0b1</td>
<td>The MPAMCFG_MBW_MAX register exists and the maximum memory bandwidth allocation scheme is supported.</td>
</tr>
</tbody>
</table>

### HAS_MIN, bit [10]

Indicates that this MSC implements minimum bandwidth partitioning.

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>There is no minimum memory bandwidth control and no MPAMCFG_MBW_MIN register.</td>
</tr>
<tr>
<td>0b1</td>
<td>The MPAMCFG_MBW_MIN register exists and the minimum memory bandwidth allocation scheme is supported.</td>
</tr>
</tbody>
</table>

### Bits [9:6]

Reserved, RES0.

### BWA_WD, bits [5:0]

Number of implemented bits in the bandwidth allocation fields: MIN, MAX and STRIDE. See MPAMCFG_MBW_MIN, MPAMCFG_MBW_MAX and MPAMCFG_MBW_PROP.

This field must have a value from 1 to 16, inclusive.
Accessing the MPAMF_MBW_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_MBW_IDR is read-only.

MPAMF_MBW_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_MBW_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_MBW_IDR_s) and Non-secure (MPAMF_MBW_IDR_ns) MPAM feature pages.

**MPAMF_MBW_IDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0040</td>
<td>MPAMF_MBW_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0040</td>
<td>MPAMF_MBW_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
MPAMF_MBWUMON_IDR, MPAM Features Memory Bandwidth Usage Monitoring ID register

The MPAMF_MBWUMON_IDR characteristics are:

Purpose

The MPAMF_MBWUMON_IDR is a 32-bit read-only register that indicates the number of memory bandwidth usage monitors for this MSC. MPAMF_MBWUMON_IDR_s indicates the number of Secure memory bandwidth usage monitor instances. MPAMF_MBWUMON_IDR_ns indicates the number of Non-secure memory bandwidth usage monitor instances.

Configuration

The power domain of MPAMF_MBWUMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MPAMF_MBWUMON_IDR are RES0.

Attributes

MPAMF_MBWUMON_IDR is a 32-bit register.

Field descriptions

The MPAMF_MBWUMON_IDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| HAS_CAPTURE | HAS_LONG | LWD | RES0 | SCALE | NUM_MON |

HAS_CAPTURE, bit [31]

The implementation of this MSC supports copying an MSMON_MBWU to the corresponding MSMON_MBWU_CAPTURE on a capture event.

<table>
<thead>
<tr>
<th>HAS_CAPTURE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSMON_MBWU_CAPTURE is not implemented and there is no support for capture events in the MBWU monitor.</td>
</tr>
<tr>
<td>0b1</td>
<td>The MSMON_MBWU_CAPTURE register is implemented and the MBWU monitor supports the capture event behavior.</td>
</tr>
</tbody>
</table>

HAS_LONG, bit [30]

When ARMv8.6-MPAM is implemented:

Indicates whether MSMON_MBWU_L is implemented.

If HAS_CAPTURE == 1, indicates whether MSMON_MBWU_L_CAPTURE is implemented.

<table>
<thead>
<tr>
<th>HAS_LONG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not implement MSMON_MBWU_L or MSMON_MBWU_L_CAPTURE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Implements MSMON_MBWU_L. If HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE is also implemented.</td>
</tr>
</tbody>
</table>
Otherwise:

Reserved, RES0.

**LWD, bit [29]**

When ARMv8.6-MPAM is implemented:

Long register VALUE width.

If MPAMF_MBWUMON_IDR.HAS_LONG == 0, MPAMF_MBWUMON_IDR.LWD must also be 0.

<table>
<thead>
<tr>
<th>LWD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>If MPAMF_MBWUMON_IDR.HAS_LONG == 1, MSMON_MBWU_L has 44-bit VALUE field in bits [43:0]. Bits [62:44] are RES0. If HAS_LONG == 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE also has 44-bit VALUE field in bits [43:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>MSMON_MBWU_L has 63-bit VALUE field in bits [62:0]. If MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE also has 63-bit VALUE field in bits [62:0].</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**Bits [28:21]**

Reserved, RES0.

**SCALE, bits [20:16]**

Scaling of MSMON_MBWU.VALUE in bits. If scaling is enabled by MSMON_CFG_MBWU_CTL.SCLEN, the byte count in the VALUE field has been shifted by SCALE bits to the right.

<table>
<thead>
<tr>
<th>SCALE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>Scaling is not implemented.</td>
</tr>
<tr>
<td>0bxxxxxx</td>
<td>Other values are right shift count when scaling is enabled.</td>
</tr>
</tbody>
</table>

**NUM_MON, bits [15:0]**

The number of memory bandwidth usage monitors implemented in this MSC.

**Accessing the MPAMF_MBWUMON_IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_MBWUMON_IDR is read-only.

MPAMF_MBWUMON_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_MBWUMON_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_MBWUMON_IDR_s) and Non-secure (MPAMF_MBWUMON_IDR_ns) MPAM feature pages.
**MPAMF_MBWUMON_IDR** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0090</td>
<td>MPAMF_MBWUMON_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0090</td>
<td>MPAMF_MBWUMON_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The MPAMF_MSMON_IDR characteristics are:

**Purpose**

The MPAMF_MSMON_IDR is a 32-bit read-only register that indicates which MPAM monitoring features are present on this MSC. MPAMF_MSMON_IDR_s indicates Secure monitoring features. MPAMF_MSMON_IDR_ns indicates Non-secure monitoring features.

**Configuration**

The power domain of MPAMF_MSMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1. Otherwise, direct accesses to MPAMF_MSMON_IDR are RES0.

**Attributes**

MPAMF_MSMON_IDR is a 32-bit register.

**Field descriptions**

The MPAMF_MSMON_IDR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| HAS_LOCAL_CAPT_EVNT | RES0 | MSMON_MBWU | MSMON_CSU | RES0 |

**HAS_LOCAL_CAPT_EVNT, bit [31]**

Has local capture event generator. Indicates whether this MSC has the MPAM local capture event generator and the MSMON_CAPT_EVNT register.

<table>
<thead>
<tr>
<th>HAS_LOCAL_CAPT_EVNT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Does not support MPAM local capture event generator or MSMON_CAPT_EVNT.</td>
</tr>
<tr>
<td>0b1</td>
<td>Supports the MPAM local capture event generator and the MSMON_CAPT_EVNT register.</td>
</tr>
</tbody>
</table>

**Bits [30:18]**

Reserved, RES0.

**MSMON_MBWU, bit [17]**

Memory bandwidth usage monitoring. Indicates whether this MSC has MPAM monitoring for Memory Bandwidth Usage by PARTID and PMG.
MSMON_MBWU

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

MSMON_CSU, bit [16]

Cache storage usage monitoring. Indicates whether this MSC has MPAM monitoring of cache storage usage by PARTID and PMG.

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
</tr>
<tr>
<td>0b1</td>
</tr>
</tbody>
</table>

Bits [15:0]

Reserved, RES0.

Accessing the MPAMF_MSMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_MSMON_IDR is read-only.

MPAMF_MSMON_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_MSMON_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_MSMON_IDR_s) and Non-secure (MPAMF_MSMON_IDR_ns) MPAM feature pages.

MPAMF_MSMON_IDR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0080</td>
<td>MPAMF_MSMON_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0080</td>
<td>MPAMF_MSMON_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
MPAMF_PARTID_NRW_IDR, MPAM PARTID Narrowing ID register

The MPAMF_PARTID_NRW_IDR characteristics are:

**Purpose**

The MPAMF_PARTID_NRW_IDR is a 32-bit read-only register that indicates the largest internal PARTID for this MSC. MPAMF_PARTID_NRW_IDR_s indicates the largest Secure internal PARTID. MPAMF_PARTID_NRW_IDR_ns indicates the largest Non-secure internal PARTID.

**Configuration**

The power domain of MPAMF_PARTID_NRW_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMF_PARTID_NRW_IDR are RES0.

**Attributes**

MPAMF_PARTID_NRW_IDR is a 32-bit register.

**Field descriptions**

The MPAMF_PARTID_NRW_IDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>INTPARTID_MAX</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**INTPARTID_MAX, bits [15:0]**

The largest intPARTID supported in this MSC.

**Accessing the MPAMF_PARTID_NRW_IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_PARTID_NRW_IDR is read-only.

MPAMF_PARTID_NRW_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_PARTID_NRW_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_PARTID_NRW_IDR_s) and Non-secure (MPAMF_PARTID_NRW_IDR_ns) MPAM feature pages.

**MPAMF_PARTID_NRW_IDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0050</td>
<td>MPAMF_PARTID_NRW_IDR_s</td>
</tr>
</tbody>
</table>
Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0050</td>
<td>MPAMF_PARTID_NRW_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
The MPAMF_PRI_IDR characteristics are:

**Purpose**

The MPAMF_PRI_IDR is a 32-bit read-only register that indicates which MPAM priority partitioning features are present on this MSC. MPAMF_PRI_IDR_s indicates priority partitioning features accessed from the Secure MPAM feature page. MPAMF_PRI_IDR_ns indicates priority partitioning features accessed from the Non-secure MPAM feature page.

**Configuration**

The power domain of MPAMF_PRI_IDR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMF_PRI_IDR are RES0.

**Attributes**

MPAMF_PRI_IDR is a 32-bit register.

**Field descriptions**

The MPAMF_PRI_IDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25-20</td>
<td>DSPRI_WD, bits [25:20]</td>
<td>Number of implemented bits in the downstream priority field (DSPRI) of MPAMCFG_PRI.</td>
</tr>
<tr>
<td>19-18</td>
<td>DSPRI_0_IS_LOW, bit [17]</td>
<td>Indicates whether 0 in MPAMCFG_PRI.DSPRI is the lowest or the highest priority.</td>
</tr>
</tbody>
</table>

**Bits [31:26]**

Reserved, RES0.

**DSPRI_WD, bits [25:20]**

Number of implemented bits in the downstream priority field (DSPRI) of MPAMCFG_PRI.

If HAS_DSPRI == 1, this field must contain a value from 1 to 32, inclusive.

If HAS_DSPRI == 0, this field must be 0.

**Bits [19:18]**

Reserved, RES0.

**DSPRI_0_IS_LOW, bit [17]**

Indicates whether 0 in MPAMCFG_PRI.DSPRI is the lowest or the highest priority.

<table>
<thead>
<tr>
<th>DSPRI_0_IS_LOW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>In the MPAMCFG_PRI.DSPRI field, a value of 0 means the highest priority.</td>
</tr>
<tr>
<td>0b1</td>
<td>In the MPAMCFG_PRI.DSPRI field, a value of 0 means the lowest priority.</td>
</tr>
</tbody>
</table>
**HAS_DSPRI, bit [16]**

Indicates that this MSC implements the DSPRI field in the MPAMCFG_PRI register.

<table>
<thead>
<tr>
<th>HAS_DSPRI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This MSC supports priority partitioning, but does not implement a downstream priority (DSPRI) field in the MPAMCFG_PRI register.</td>
</tr>
<tr>
<td>0b1</td>
<td>This MSC supports downstream priority partitioning and implements the downstream priority (DSPRI) field in the MPAMCFG_PRI register.</td>
</tr>
</tbody>
</table>

**Bits [15:10]**

Reserved, RES0.

**INTPRI_WD, bits [9:4]**

Number of implemented bits in the internal priority field (INTPRI) in the MPAMCFG_PRI register.

If HAS_INTPRI == 1, this field must contain a value from 1 to 32, inclusive.

If HAS_INTPRI == 0, this field must be 0.

**Bits [3:2]**

Reserved, RES0.

**INTPRI_0_IS_LOW, bit [1]**

Indicates whether 0 in MPAMCFG_PRI.INTPRI is the lowest or the highest priority.

<table>
<thead>
<tr>
<th>INTPRI_0_IS_LOW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>In the MPAMCFG_PRI. INTPRI field, a value of 0 means the highest priority.</td>
</tr>
<tr>
<td>0b1</td>
<td>In the MPAMCFG_PRI. INTPRI field, a value of 0 means the lowest priority.</td>
</tr>
</tbody>
</table>

**HAS_INTPRI, bit [0]**

Indicates that this MSC implements the INTPRI field in the MPAMCFG_PRI register.

<table>
<thead>
<tr>
<th>HAS_INTPRI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This MSC supports priority partitioning, but does not implement the internal priority (INTPRI) field in the MPAMCFG_PRI register.</td>
</tr>
<tr>
<td>0b1</td>
<td>This MSC supports internal priority partitioning and implements the internal priority (INTPRI) field in the MPAMCFG_PRI register.</td>
</tr>
</tbody>
</table>

**Accessing the MPAMF_PRI_IDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_PRI_IDR is read-only.

MPAMF_PRI_IDR must be readable from the Non-secure and Secure MPAM feature pages.

MPAMF_PRI_IDR is permitted to have the same contents when read from either the Secure and Non-secure MPAM feature pages unless the register contents is different for Secure and Non-secure versions, when there must be separate registers in the Secure (MPAMF_PRI_IDR_s) and Non-secure (MPAMF_PRI_IDR_ns) MPAM feature pages.
**MPAMF_PRI_IDR** can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0048</td>
<td>MPAMF_PRI_IDR_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0048</td>
<td>MPAMF_PRI_IDR_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RO**.
MPAMF_SIDR, MPAM Features Secure Identification Register

The MPAMF_SIDR characteristics are:

**Purpose**

The MPAMF_SIDR is a 32-bit read-only register that indicates the maximum Secure PARTID and Secure PMG on this MSC.

**Configuration**

The power domain of MPAMF_SIDR is IMPLEMENTATION DEFINED.

**Attributes**

MPAMF_SIDR is a 32-bit register.

**Field descriptions**

The MPAMF_SIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>23-16</td>
<td>S_PMG_MAX</td>
</tr>
<tr>
<td>15-0</td>
<td>S_PARTID_MAX</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**S_PMG_MAX, bits [23:16]**

Maximum value of Secure PMG supported by this component.

**S_PARTID_MAX, bits [15:0]**

Maximum value of Secure PARTID supported by this component.

**Accessing the MPAMF_SIDR**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMF_SIDR is read-only.

MPAMF_SIDR must only be readable from the Secure MPAM feature page. If the system or the MSC does not support the Secure address map, this register must not be accessible.

**MPAMF_SIDR can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE</td>
<td>0x0008</td>
<td>MPAMF_SIDR</td>
</tr>
</tbody>
</table>

Accesses on this interface are RO.
MSMON_CAPT_EVNT, MPAM Capture Event Generation Register

The MSMON_CAPT_EVNT characteristics are:

**Purpose**

MSMON_CAPT_EVNT is a 32-bit read-write register that generates a local capture event when written with bit[0] as 1. MSMON_CAPT_EVNT_s generates local capture events for Secure monitors only or for Secure and Non-secure monitors. MSMON_CAPT_EVNT_ns generates local capture events for Non-secure monitors only.

**Configuration**

The power domain of MSMON_CAPT_EVNT is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1. Otherwise, direct accesses to MSMON_CAPT_EVNT are RES0.

**Attributes**

MSMON_CAPT_EVNT is a 32-bit register.

**Field descriptions**

The MSMON_CAPT_EVNT bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Send capture event to Secure monitors only.</td>
</tr>
<tr>
<td>30</td>
<td>Send capture event to both Secure and Non-secure monitors.</td>
</tr>
</tbody>
</table>

**Bits [31:2]**

Reserved, RES0.

**ALL, bit [1]**

In the Secure instance of this register, if ALL written as 1 and NOW is also written as 1, signal a capture event to Secure and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

If written as 0 and NOW is written as 1, signal a capture event to Secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

In the Non-secure instance of this register, this bit is RAZ/WI.

This bit always reads as zero.

<table>
<thead>
<tr>
<th>ALL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Send capture event to Secure monitors only.</td>
</tr>
<tr>
<td>0b1</td>
<td>Send capture event to both Secure and Non-secure monitors.</td>
</tr>
</tbody>
</table>

**NOW, bit [0]**

When written as 1, this bit causes an event to all monitors in this MSC with CAPT_EVNT set to the value of 7.

When this bit is written as 0, no event is signaled.

This bit always reads as zero.
**Accessing the MSMON\_CAPT\_EVNT**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON\_CAPT\_EVNT\_s must be accessible from the Secure MPAM feature page. MSMON\_CAPT\_EVENT\_ns must be accessible from the Non-secure MPAM feature page.

The two instances of MSMON\_CAPT\_EVNT must be separate registers. The Secure instance (MSMON\_CAPT\_EVNT\_s) can generate capture events for both Secure and Non-secure PARTID monitors, and the Non-secure instance (MSMON\_CAPT\_EVNT\_ns) can generate capture events for Non-secure PARTID monitors only.

**MSMON\_CAPT\_EVNT can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0808</td>
<td>MSMON_CAPT_EVNT_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0808</td>
<td>MSMON_CAPT_EVNT_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbfe3431cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

The MSMON_CFG_CSU_CTL characteristics are:

**Purpose**

MSMON_CFG_CSU_CTL is a 32-bit read-write register that controls the CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL controls the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL_ns controls Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_CFG_CSU_CTL is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_CTL are RES0.

**Attributes**

MSMON_CFG_CSU_CTL is a 32-bit register.

**Field descriptions**

The MSMON_CFG_CSU_CTL bit assignments are:

<table>
<thead>
<tr>
<th>EN</th>
<th>CAPT_EVNT</th>
<th>CAPT_RESET</th>
<th>OFLOW_STATUS</th>
<th>OFLOW_INTR</th>
<th>OFLOW_FRZ</th>
<th>SUBTYPE</th>
<th>RES0</th>
<th>MATCH_PMG</th>
<th>MATCH_PARTID</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>30</td>
<td>29</td>
<td>28</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>24</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>20</td>
<td>19</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>15</td>
<td>14</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**EN, bit [31]**

Enabled.

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor instance is disabled and must not collect any information.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor instance is enabled to collect information according to the configuration of the instance.</td>
</tr>
</tbody>
</table>

**CAPT_EVNT, bits [30:28]**

Capture event selector.

Select the event that triggers capture from the following:
<table>
<thead>
<tr>
<th>CAPT_EVNT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>No capture event is triggered.</td>
</tr>
<tr>
<td>0b001</td>
<td>External capture event 1 (optional but recommended)</td>
</tr>
<tr>
<td>0b010</td>
<td>External capture event 2 (optional)</td>
</tr>
<tr>
<td>0b011</td>
<td>External capture event 3 (optional)</td>
</tr>
<tr>
<td>0b100</td>
<td>External capture event 4 (optional)</td>
</tr>
<tr>
<td>0b101</td>
<td>External capture event 5 (optional)</td>
</tr>
<tr>
<td>0b110</td>
<td>External capture event 6 (optional)</td>
</tr>
<tr>
<td>0b111</td>
<td>Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the security state of this monitor. (optional)</td>
</tr>
</tbody>
</table>

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources should not trigger a capture event.

If capture is not implemented for the CSU monitor type as indicated by `MPAMF_CSUMON_IDR.HAS_CAPTURE = 0`, this field is RAZ/WI.

**CAPT_RESET, bit [27]**

Reset after capture.

Controls whether the value of `MSMON_CSU` is reset to zero immediately after being copied to `MSMON_CSU_CAPTURE`.

<table>
<thead>
<tr>
<th>CAPT_RESET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Monitor is not reset on capture.</td>
</tr>
<tr>
<td>0b1</td>
<td>Monitor is reset on capture.</td>
</tr>
</tbody>
</table>

If capture is not implemented for the CSU monitor type as indicated by `MPAMF_CSUMON_IDR.HAS_CAPTURE = 0`, this field is RAZ/WI.

Because the CSU monitor type produces a measurement rather than a count, it might not make sense to ever reset the value after a capture. If there is no reason to ever reset a CSU monitor, this field is RAZ/WI.

**OFLOW_STATUS, bit [26]**

Overflow status.

Indicates whether the value of `MSMON_CSU` has overflowed.

<table>
<thead>
<tr>
<th>OFLOW_STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No overflow has occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one overflow has occurred since this bit was last written to zero.</td>
</tr>
</tbody>
</table>

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

**OFLOW_INTR, bit [25]**

Overflow Interrupt.

Controls whether an overflow interrupt is generated when the value of `MSMON_CSU` has overflowed.

<table>
<thead>
<tr>
<th>OFLOW_INTR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt is signaled on an overflow of <code>MSMON_CSU</code>.</td>
</tr>
<tr>
<td>0b1</td>
<td>On overflow, an implementation-specific interrupt is signaled.</td>
</tr>
</tbody>
</table>

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

**OFLOW_FRZ, bit [24]**

Freeze Monitor on Overflow.
Controls whether the value of \texttt{MSMON\_CSU} freezes on an overflow.

<table>
<thead>
<tr>
<th>\texttt{OFLOW_FRZ}</th>
<th>\textbf{Meaning}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Monitor count wraps on overflow.</td>
</tr>
<tr>
<td>0b1</td>
<td>Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1.</td>
</tr>
</tbody>
</table>

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

\textbf{SUBTYPE, bits [23:20]}

Subtype.

A monitor can have other event matching criteria.

This field is not currently used for CSU monitors, but reserved for future use.

This field is RAZ/WI.

\textbf{Bits [19:18]}

Reserved, RES0.

\textbf{MATCH\_PMG, bit [17]}

Match PMG.

Controls whether the monitor measures only storage used with PMG matching \texttt{MSMON\_CFG\_CSU\_FLT.PMG}.

<table>
<thead>
<tr>
<th>\texttt{MATCH_PMG}</th>
<th>\textbf{Meaning}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor measures storage used with any PMG value.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor only measures storage used with the PMG value matching \texttt{MSMON_CFG_CSU_FLT.PMG}.</td>
</tr>
</tbody>
</table>

If MATCH\_PMG == 1 and MATCH\_PARTID == 0, it is \texttt{CONSTRAINED\_UNPREDICTABLE} whether the monitor instance:

- Measures the storage used with matching PMG and with any PARTID.
- Measures no storage usage, that is, \texttt{MSMON\_CSU.VALUE} is zero.
- Measures the storage used with matching PMG and PARTID, that is, treats MATCH\_PARTID as == 1.

\textbf{MATCH\_PARTID, bit [16]}

Match PARTID.

Controls whether the monitor measures only storage used with PARTID matching \texttt{MSMON\_CFG\_CSU\_FLT.PARTID}.

<table>
<thead>
<tr>
<th>\texttt{MATCH_PARTID}</th>
<th>\textbf{Meaning}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor measures storage used with any PARTID value.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor only measures storage used with the PARTID value matching \texttt{MSMON_CFG_CSU_FLT.PARTID}.</td>
</tr>
</tbody>
</table>

\textbf{Bits [15:8]}

Reserved, RES0.

\textbf{TYPE, bits [7:0]}

Monitor Type Code.

Constant type indicating the type of the monitor.
CSU monitor is $\text{TYPE} = 0x43$.

## Accessing the MSMON_CFG_CSU_CTL

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_CSU_CTL_s must be accessible from the Secure MPAM feature page. MSMON_CFG_CSU_CTL_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_CSU_CTL_s and MSMON_CFG_CSU_CTL_ns must be separate registers. The Secure instance (MSMON_CFG_CSU_CTL_s) accesses the cache storage usage monitor controls used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_CSU_CTL_ns) accesses the cache storage usage monitor controls used for Non-secure PARTIDs.

### MSMON_CFG_CSU_CTL can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0818</td>
<td>MSMON_CFG_CSU_CTL_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0818</td>
<td>MSMON_CFG_CSU_CTL_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

The MSMON_CFG_CSU_FLT characteristics are:

**Purpose**

MSMON_CFG_CSU_FLT is a 32-bit read-write register that sets PARTID and PMG to measure or count in the CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CFG_CSU_FLT sets filter conditions for the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_CSU_CTL sets filter conditions for the Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_CFG_CSU_FLT is IMPLEMENTATION DEFINED.

This register is present only when MPMF_IDR.HAS_MSMON == 1 and MPMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_FLT are RES0.

**Attributes**

MSMON_CFG_CSU_FLT is a 32-bit register.

**Field descriptions**

The MSMON_CFG_CSU_FLT bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>PMG</td>
</tr>
<tr>
<td>29</td>
<td>PARTID</td>
</tr>
</tbody>
</table>

**Bits [31:24]**

Reserved, RES0.

**PMG, bits [23:16]**

Performance monitoring group to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 0, this field is not used to match cache storage to a PMG and the contents of this field is ignored.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID == 1, the monitor instance selected by MSMON_CFG_MON_SEL measures or counts cache storage labeled with PMG equal to this field and PARTID equal to the PARTID field.

If MSMON_CFG_CSU_CTL.MATCH_PMG == 1 and MSMON_CFG_CSU_CTL.MATCH_PARTID == 0, the behavior of the monitor instance selected by MSMON_CFG_MON_SEL is CONSTRAINED UNPREDICTABLE. See MSMON_CFG_CSU_CTL.MATCH_PMG for more information.

**PARTID, bits [15:0]**

Partition ID to filter cache storage usage monitoring.

If MSMON_CFG_CSU_CTL.MATCH_PARTID == 0 and MSMON_CFG_CSU_CTL.MATCH_PMG == 0, the monitor measures all allocated cache storage.
If `MSMON_CFG_CSU_CTL.MATCH_PARTID == 0` and `MSMON_CFG_CSU_CTL.MATCH_PMG == 1`, the behavior of the monitor is **CONSTRAINED UNPREDICTABLE**. See the description of `MSMON_CFG_CSU_CTL.MATCH_PMG`.

If `MSMON_CFG_CSU_CTL.MATCH_PARTID == 1` and `MSMON_CFG_CSU_CTL.MATCH_PMG == 0`, the monitor selected by `MSMON_CFG_MON_SEL` measures or counts cache storage labeled with PARTID equal to this field.

If `MSMON_CFG_CSU_CTL.MATCH_PARTID == 1` and `MSMON_CFG_CSU_CTL.MATCH_PMG == 1`, the monitor selected by `MSMON_CFG_MON_SEL` measures or counts cache storage labeled with PARTID equal to this field and PMG equal to the PMG field.

### Accessing the MSMON_CFG_CSU_FLT

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

`MSMON_CFG_CSU_FLT_s` must be accessible from the Secure MPAM feature page. `MSMON_CFG_CSU_FLT_ns` must be accessible from the Non-secure MPAM feature page.

`MSMON_CFG_CSU_FLT_s` and `MSMON_CFG_CSU_FLT_ns` must be separate registers. The Secure instance (`MSMON_CFG_CSU_FLT_s`) accesses the PARTID and PMG matching for a cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance (`MSMON_CFG_CSU_FLT_ns`) accesses the PARTID and PMG matching for a cache storage usage monitor used for Non-secure PARTIDs.

**MSMON_CFG_CSU_FLT can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0810</td>
<td>MSMON_CFG_CSU_FLT_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0810</td>
<td>MSMON_CFG_CSU_FLT_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MSMON_CFG_MBWU_CTL characteristics are:

**Purpose**

Controls the MBWU monitor selected by \texttt{MSMONCFG_MON_SEL}. \texttt{MSMONCFG_MBWU_CTL} controls the Secure memory bandwidth usage monitor instance selected by the Secure instance of \texttt{MSMONCFG_MON_SEL}. \texttt{MSMONCFG_MBWU_CTL}_{ns} controls Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of \texttt{MSMONCFG_MON_SEL}.

**Configuration**

The power domain of MSMON_CFG_MBWU_CTL is \texttt{IMPLEMENTATION DEFINED}.

This register is present only when \texttt{MPAMF_IDR.HAS_MSMON == 1} and \texttt{MPAMF_MSMON_IDR.MSMON_MBWU == 1}. Otherwise, direct accesses to MSMON_CFG_MBWU_CTL are \texttt{RES0}.

**Attributes**

MSMON_CFG_MBWU_CTL is a 32-bit register.

**Field descriptions**

The MSMON_CFG_MBWU_CTL bit assignments are:

<table>
<thead>
<tr>
<th>EN</th>
<th>CAPT_EVNT</th>
<th>CAPT_RESET</th>
<th>OFLOW_STATUS</th>
<th>OFLOW_INTR</th>
<th>OFLOW_FRZ</th>
<th>SUBTYPE</th>
<th>SCLEN</th>
<th>RES0</th>
<th>MATCH_PMG</th>
<th>MATCH_PART</th>
</tr>
</thead>
</table>

**EN, bit [31]**

Enabled.

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor instance is disabled and must not collect any information.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor instance is enabled to collect information according to the configuration of the instance.</td>
</tr>
</tbody>
</table>

**CAPT_EVNT, bits [30:28]**

Capture event selector.

When the selected capture event occurs, \texttt{MSMON_MBWU} of the monitor instance is copied to \texttt{MSMON_MBWU_CAPTURE} of the same instance. If the long counter is also implemented, \texttt{MSMON_MBWU_L} is also copied to \texttt{MSMON_MBWU_L_CAPTURE}.

Select the event that triggers capture from the following:
The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources should not trigger a capture event.

If capture is not implemented for the MBWU monitor type as indicated by MPAMF_MBWUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

**CAPT_RESET, bit [27]**

Reset MSMON_MBWU.VALUE after capture.

Controls whether the VALUE field of the monitor instance is reset to zero immediately after being copied to the corresponding capture register.

<table>
<thead>
<tr>
<th>CAPT_RESET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSMON_MBWU.VALUE field of the monitor instance is not reset on capture.</td>
</tr>
<tr>
<td>0b1</td>
<td>MSMON_MBWU.VALUE field of the monitor instance is reset on capture.</td>
</tr>
</tbody>
</table>

If capture is not implemented for the MBWU monitor type as indicated by MPAMF_MBWUMON_IDR.HAS_CAPTURE = 0, this field is RAZ/WI.

This control bit affects both MSMON_MBWU and MSMON_MBWU_L in implementations that include MSMON_MBWU_L.

**OFLOW_STATUS, bit [26]**

Overflow status.

Indicates whether the value of MSMON_MBWU has overflowed.

<table>
<thead>
<tr>
<th>OFLOW_STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSMON_MBWU has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>MSMON_MBWU has overflowed at least once since this bit was last written to zero.</td>
</tr>
</tbody>
</table>

If overflow is not possible for a MBWU monitor in the implementation, this field is RA/Z/WI.

**OFLOW_INTR, bit [25]**

Overflow status of the MSMON_MBWU_L instance.

Indicates that the value of MSMON_MBWU overflow has overflowed.

<table>
<thead>
<tr>
<th>OFLOW_INTR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt is signaled on an overflow of MSMON_MBWU.</td>
</tr>
<tr>
<td>0b1</td>
<td>An implementation-specific interrupt is signaled on an overflow of MSMON_MBWU.</td>
</tr>
</tbody>
</table>

**OFLOW_FRZ, bit [24]**

Freeze monitor instance on overflow.
Controls whether \texttt{MSMON\_MBWU\_VALUE} field of the monitor instance freezes on an overflow.

<table>
<thead>
<tr>
<th>OFLOW_FRZ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>\texttt{MSMON_MBWU_VALUE} field of the monitor instance wraps on overflow.</td>
</tr>
<tr>
<td>0b1</td>
<td>\texttt{MSMON_MBWU_VALUE} field of the monitor instance freezes on overflow. If the increment that caused the overflow was 1, the frozen value is the post-increment value of 0. If the increment that caused the overflow was larger than 1, the frozen value of the monitor might be 0 or a larger value less than the final increment.</td>
</tr>
</tbody>
</table>

If overflow is not possible for the instance of the MBWU monitor in the implementation, this field is RAZ/WI.

This control bit affects both \texttt{MSMON\_MBWU} and \texttt{MSMON\_MBWU\_L} in implementations that include \texttt{MSMON\_MBWU\_L}.

**SUBTYPE, bits [23:20]**

Subtype.

A monitor can have other event matching criteria.

This field is not currently used for MBWU monitors, but reserved for future use.

This field is RAZ/WI.

**SCLEN, bit [19]**

\texttt{MSMON\_MBWU\_VALUE} Scaling Enable.

Enables scaling of \texttt{MSMON\_MBWU\_VALUE} by \texttt{MPAMF\_MBWUMON\_IDR\_SCALE}.

<table>
<thead>
<tr>
<th>SCLEN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>\texttt{MSMON_MBWU_VALUE} has bytes counted by the monitor instance.</td>
</tr>
<tr>
<td>0b1</td>
<td>\texttt{MSMON_MBWU_VALUE} has bytes counted by the monitor instance, shifted right by \texttt{MPAMF_MBWUMON_IDR_SCALE}.</td>
</tr>
</tbody>
</table>

**Bit [18]**

Reserved, RES0.

**MATCH\_PMG, bit [17]**

Match PMG.

Controls whether the monitor instance only counts data transferred with PMG matching \texttt{MSMON\_CFG\_MBWU\_FLT\_PMG}.

<table>
<thead>
<tr>
<th>MATCH_PMG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor instance counts data transferred with any PMG value.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor instance only counts data transferred with the PMG value matching \texttt{MSMON_CFG_MBWU_FLT_PMG}.</td>
</tr>
</tbody>
</table>

**MATCH\_PARTID, bit [16]**

Match PARTID.

Controls whether the monitor instance counts only data transferred with PARTID matching \texttt{MSMON\_CFG\_MBWU\_FLT\_PARTID}. 
<table>
<thead>
<tr>
<th>MATCH_PARTID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor instance counts data transferred with any PARTID value.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor instance only counts data transferred with the PARTID value matching MSMON_CFG_MBWU_FLT.PARTID.</td>
</tr>
</tbody>
</table>

**OFLOW_STATUS_L, bit [15]**

When ARMv8.6-MPAM is implemented:

Overflow Status of the MSMON_MBWU_L instance.

Indicates whether an MPAM overflow interrupt is generated when the value of MSMON_MBWU_L has overflowed.

<table>
<thead>
<tr>
<th>OFLOW_STATUS_L</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>MSMON_MBWU_L has not overflowed.</td>
</tr>
<tr>
<td>0b1</td>
<td>MSMON_MBWU_L has overflowed at least once since this bit was last written to zero.</td>
</tr>
</tbody>
</table>

If MPAMF_MBWUMON_IDR.HAS_LONG == 0, this bit is RES0.

Otherwise:

Reserved, RES0.

**OFLOW_INTR_L, bit [14]**

When ARMv8.6-MPAM is implemented:

Overflow Interrupt for MSMON_MBWU_L.

Controls whether an MPAM overflow interrupt is generated when the value of MSMON_MBWU_L overflows.

<table>
<thead>
<tr>
<th>OFLOW_INTR_L</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No interrupt is signaled on an overflow of MSMON_MBWU_L.</td>
</tr>
<tr>
<td>0b1</td>
<td>An implementation-specific interrupt is signalled on overflow of MSMON_MBWU_L.</td>
</tr>
</tbody>
</table>

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

If MPAMF_MBWUMON_IDR.HAS_LONG == 0, this bit is RES0.

Otherwise:

Reserved, RES0.

**Bits [13:8]**

Reserved, RES0.

**TYPE, bits [7:0]**

Monitor Type Code.

Constant type indicating the type of the monitor.

Read-only.

MBWU monitor is TYPE = 0x42.
## Accessing the MSMON_CFG_MBWU_CTL

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_MBWU_CTL_s must be accessible from the Secure MPAM feature page. MSMON_CFG_MBWU_CTL_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_MBWU_CTL_s and MSMON_CFG_MBWU_CTL_ns must be separate registers. The Secure instance (MSMON_CFG_MBWU_CTL_s) accesses the memory bandwidth usage monitor controls used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_MBWU_CTL_ns) accesses the memory bandwidth usage monitor controls used for Non-secure PARTIDs.

**MSMON_CFG_MBWU_CTL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0828</td>
<td>MSMON_CFG_MBWU_CTL_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0828</td>
<td>MSMON_CFG_MBWU_CTL_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
The MSMON_CFG_MBWU_FLT characteristics are:

**Purpose**

MSMON_CFG_MBWU_FLT is a 32-bit read-write register that sets PARTID and PMG to measure or count in the MBWU monitor selected by MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_FLT sets filter conditions for the Secure memory bandwidth usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CFG_MBWU_CTL sets filter conditions for the Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_CFG_MBWU_FLT is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_FLT are RES0.

**Attributes**

MSMON_CFG_MBWU_FLT is a 32-bit register.

**Field descriptions**

The MSMON_CFG_MBWU_FLT bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | PMG | PARTID |

**Bits [31:24]**

Reserved, RES0.

**PMG, bits [23:16]**

Performance monitoring group to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 0, this field is not used to match memory bandwidth to a PMG and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PMG == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PMG equal to this field.

**PARTID, bits [15:0]**

Partition ID to filter memory bandwidth usage monitoring.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 0, this field is not used to match memory bandwidth to a PARTID and the contents of this field is ignored.

If MSMON_CFG_MBWU_CTL.MATCH_PARTID == 1, the monitor selected by MSMON_CFG_MON_SEL measures or counts memory bandwidth labeled with PARTID equal to this field.
Accessing the MSMON_CFG_MBWU_FLT

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CFG_MBWU_FLT_s must be accessible from the Secure MPAM feature page. MSMON_CFG_MBWU_FLT_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CFG_MBWU_FLT_s and MSMON_CFG_MBWU_FLT_ns must be separate registers. The Secure instance (MSMON_CFG_MBWU_FLT_s) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CFG_MBWU_FLT_ns) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_CFG_MBWU_FLT can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0820</td>
<td>MSMON_CFG_MBWU_FLT_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0820</td>
<td>MSMON_CFG_MBWU_FLT_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are RW.
MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register

The MSMON_CFG_MON_SEL characteristics are:

**Purpose**

Selects a monitor instance to access through the MSMON configuration and counter registers. MSMON_CFG_MON_SEL selects a Secure monitor instance to access via the Secure MPAM feature page. MSMON_CFG_MON_SEL_ns selects a Non-secure monitor instance to access via the Non-secure MPAM feature page.

**Note**

Different performance monitoring features within a MSC could have different numbers of monitor instances. See the NUM_MON field in the corresponding ID register. This means that a monitor out-of-bounds error might be signaled when an MSMON_CFG register is accessed because the value in MSMON_CFG_MON_SEL.MON_SEL is too large for the particular monitoring feature.

To configure a monitor, set MON_SEL in this register to the index of the monitor instance to configure, then write to the MSMON_CFG_x register to set the configuration of the monitor. At a later time, read the monitor register (for example MSMON_MBWU) to get the value of the monitor.

**Configuration**

The power domain of MSMON_CFG_MON_SEL is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1, or (MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.EXT == 0) or (MPAMF_HAS_IMPL_IDR == 1, MPAMF_IDR.EXT == 1 and MPAMF_IDR.NO_IMPL_MSMON == 1). Otherwise, direct accesses to MSMON_CFG_MON_SEL are RES0.

**Attributes**

MSMON_CFG_MON_SEL is a 32-bit register.

**Field descriptions**

The MSMON_CFG_MON_SEL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>MOD, bits [28]</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>RIS, bits [24]</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [31:28]**

Reserved, RES0.

**RIS, bits [27:24]**

When ARMv8.6-MPAM is implemented, MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MSMON_CFG registers.

Otherwise:

Reserved, RES0.
Bits [23:16]

Reserved, RES0.

**MON_SEL, bits [15:0]**

Selects the monitor instance to configure or read.

Reads and writes to other MSMON registers are indexed by MON_SEL and by the NS bit used to access MSMON_CFG_MON_SEL to access the configuration for a single monitor.

**Accessing the MSMON_CFG_MON_SEL**

This register is part of the MPAMF_BASE memory frame. In a system that supports Secure and Non-secure memory maps, the MPAMF_BASE frame must be accessible in both Secure and Non-secure memory address maps.

MSMON_CFG_MON_SEL must be accessible from the Non-secure and Secure address maps.

MSMON_CFG_MON_SEL must be banked for the Secure and Non-secure address maps. The Secure instance is used with accesses to other MSMON registers to configure monitors for Secure PARTIDs, and the Non-secure instance is used with accesses to other MSMON registers to configure monitors for Non-secure PARTIDs.

**MSMON_CFG_MON_SEL can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0800</td>
<td>MSMON_CFG_MON_SEL_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0000</td>
<td>MSMON_CFG_MON_SEL_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

27/09/2019 18:23; 6134483bd14dc8c12a99c984cbe34331cc1c9707

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The MSMON_CSU characteristics are:

**Purpose**

MSMON_CSU is a 32-bit read-write register that accesses the CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CSU_s is the Secure cache storage usage monitor instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_ns is the Non-secure cache storage usage monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_CSU is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CSU are RES0.

**Attributes**

MSMON_CSU is a 32-bit register.

**Field descriptions**

The MSMON_CSU bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NRDY | VALUE |

**NRDY, bit [31]**

Not Ready. Indicates whether the monitor has possibly inaccurate data.

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor is ready and the VALUE fields is accurate.</td>
</tr>
</tbody>
</table>

**VALUE, bits [30:0]**

Cache storage usage value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the cache storage usage in bytes meeting the criteria set in MSMON_CFG_CSU_FLT and MSMON_CFG_CSU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

**Accessing the MSMON_CSU**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CSU_s must be accessible from the Secure MPAM feature page. MSMON_CSU_ns must be accessible from the Non-secure MPAM feature page.
MSMON_CSU_s and MSMON_CSU_ns must be separate registers. The Secure instance (MSMON_CSU_s) accesses the cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CSU_ns) accesses the cache storage usage monitor used for Non-secure PARTIDs.

**MSMON_CSU can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0840</td>
<td>MSMON_CSU_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0840</td>
<td>MSMON_CSU_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MSMON_CSU_CAPTURE, MPAM Cache Storage Usage Monitor Capture Register

The MSMON_CSU_CAPTURE characteristics are:

Purpose

MSMON_CSU_CAPTURE is a 32-bit read-write register that accesses the captured MSMON_CSU monitor selected by MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_s is the Secure cache storage usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_CSU_CAPTURE_ns is the Non-secure cache storage usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

Configuration

The power domain of MSMON_CSU_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_CSU == 1 and MPAMF_CSUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_CSU_CAPTURE are RES0.

Attributes

MSMON_CSU_CAPTURE is a 32-bit register.

Field descriptions

The MSMON_CSU_CAPTURE bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NRDY | VALUE |

NRDY, bit [31]

Not Ready. Indicates whether the captured monitor value has possibly inaccurate data.

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The captured monitor was not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual cache storage usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The captured monitor was ready and the VALUE fields is accurate.</td>
</tr>
</tbody>
</table>

VALUE, bits [30:0]

Captured cache storage usage value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the captured cache storage usage in bytes meeting the criteria set in MSMON_CFG_CSU_FLT and MSMON_CFG_CSU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_CSU_CAPTURE

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CSU_CAPTURE_s must be accessible from the Secure MPAM feature page. MSMON_CSU_CAPTURE_ns must be accessible from the Non-secure MPAM feature page.
MSMON_CSU_CAPTURE_s and MSMON_CSU_CAPTURE_ns must be separate registers. The Secure instance (MSMON_CSU_CAPTURE_s) accesses the captured cache storage usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_CSU_CAPTURE_ns) accesses the captured cache storage usage monitor used for Non-secure PARTIDs.

**MSMON_CSU_CAPTURE can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0848</td>
<td>MSMON_CSU_CAPTURE_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0848</td>
<td>MSMON_CSU_CAPTURE_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register

The MSMON_MBWU characteristics are:

**Purpose**

MSMON_MBWU is a 32-bit read-write register that accesses the monitor instance selected by MSMON_CFG_MON_SEL. MSMON_MBWU_s is the Secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_s. MSMON_MBWU_ns is the Non-secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_ns.

**Configuration**

The power domain of MSMON_MBWU is implementation defined.

This register is present only when MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_MBWU are RESET.

**Attributes**

MSMON_MBWU is a 32-bit register.

**Field descriptions**

The MSMON_MBWU bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| NRDY | VALUE |

**NRDY, bit [31]**

Not Ready. Indicates whether the monitor has possibly inaccurate data.

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor is ready and the VALUE fields is accurate.</td>
</tr>
</tbody>
</table>

**VALUE, bits [30:0]**

Memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the scaled count of bytes transferred since the monitor was last reset that meet the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

If MSMON_CFG_MBWU_CTL.SCLEN == 1, the count in VALUE is the number of bytes shifted right by MPAMF_MBWUMON_IDR.SCALE bit positions and rounded.

**Accessing the MSMON_MBWU**

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.
MSMON_MBWU_s must be accessible from the Secure MPAM feature page. MSMON_MBWU_ns must be accessible from the Non-secure MPAM feature page.

MSMON_MBWU_s and MSMON_MBWU_ns must be separate registers. The Secure instance (MSMON_MBWU_s) accesses the memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_MBWU_ns) accesses the memory bandwidth usage monitor used for Non-secure PARTIDs.

**MSMON_MBWU can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0860</td>
<td>MSMON_MBWU_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0860</td>
<td>MSMON_MBWU_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
MSMON_MBWU__L_CAPTURE, MPAM Long Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU__L_CAPTURE characteristics are:

**Purpose**

MSMON_MBWU__L_CAPTURE is a 64-bit read-write register that accesses the captured MSMON_MBWU instance selected by MSMON_CFG_MON_SEL.

MSMON_MBWU__L_CAPTURE_s is the Secure memory bandwidth usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL.

MSMON_MBWU__L_CAPTURE_ns is the Non-secure memory bandwidth usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_MBWU__L_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1 and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU__L_CAPTURE are IMPLEMENTATION DEFINED.

**Attributes**

MSMON_MBWU__L_CAPTURE is a 64-bit register.

**Field descriptions**

The MSMON_MBWU__L_CAPTURE bit assignments are:

**When MPAMF_MBWUMON_IDR.LWD == 0:**

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>NRDY</td>
</tr>
<tr>
<td>62</td>
<td>RES0</td>
</tr>
<tr>
<td>31</td>
<td>VALUE</td>
</tr>
</tbody>
</table>

NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor is not ready and the contents of the</td>
</tr>
<tr>
<td></td>
<td>MSMON_MBWU__L_CAPTURE.VALUE field might be inaccurate</td>
</tr>
<tr>
<td></td>
<td>or otherwise not represent the actual memory bandwidth usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor is ready and the</td>
</tr>
<tr>
<td></td>
<td>MSMON_MBWU__L_CAPTURE.VALUE field is accurate.</td>
</tr>
</tbody>
</table>

Bits [62:44]

Reserved, RES0.
VALUE, bits [43:0]

Memory bandwidth usage counter value if \texttt{MSMON\_MBWU\_L\_CAPTURE.NRDY} == 0. Invalid if \texttt{MSMON\_MBWU\_L\_CAPTURE.NRDY} == 1.

\texttt{MSMON\_MBWU\_L\_CAPTURE.VALUE} is the 44-bit count of bytes transferred since the monitor was last reset that meet the criteria set in \texttt{MSMON\_CFG\_MBWU\_FLT} and \texttt{MSMON\_CFG\_MBWU\_CTL} for the monitor instance selected by \texttt{MSMON\_CFG\_MON\_SEL}.

When \texttt{MPAMF\_MBWUMON\_IDR.LWD} == 1:

<table>
<thead>
<tr>
<th>NRDY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
</tr>
<tr>
<td>28</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>5</td>
</tr>
<tr>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>24</td>
<td>7</td>
</tr>
<tr>
<td>23</td>
<td>8</td>
</tr>
<tr>
<td>22</td>
<td>9</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>11</td>
</tr>
<tr>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>18</td>
<td>13</td>
</tr>
<tr>
<td>17</td>
<td>14</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>17</td>
</tr>
<tr>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>12</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>21</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
</tr>
<tr>
<td>8</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>27</td>
</tr>
<tr>
<td>3</td>
<td>28</td>
</tr>
<tr>
<td>2</td>
<td>29</td>
</tr>
<tr>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>31</td>
</tr>
</tbody>
</table>

NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor is not ready and the contents of the \texttt{MSMON_MBWU_L_CAPTURE.VALUE} field might be inaccurate or otherwise not represent the actual memory bandwidth usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor is ready and the \texttt{MSMON_MBWU_L_CAPTURE.VALUE} fields is accurate.</td>
</tr>
</tbody>
</table>

VALUE, bits [62:0]

Memory bandwidth usage counter value if \texttt{MSMON\_MBWU\_L\_CAPTURE.NRDY} == 0. Invalid if \texttt{MSMON\_MBWU\_L\_CAPTURE.NRDY} == 1.

\texttt{MSMON\_MBWU\_L\_CAPTURE.VALUE} is the 63-bit count of bytes transferred since the monitor was last reset that meet the criteria set in \texttt{MSMON\_CFG\_MBWU\_FLT} and \texttt{MSMON\_CFG\_MBWU\_CTL} for the monitor instance selected by \texttt{MSMON\_CFG\_MON\_SEL}.

Accessing the MSMON\_MBWU\_L\_CAPTURE

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

\texttt{MSMON\_MBWU\_L\_CAPTURE.s} must be accessible from the Secure MPAM feature page.
\texttt{MSMON\_MBWU\_L\_CAPTURE.ns} must be accessible from the Non-secure MPAM feature page.

\texttt{MSMON\_MBWU\_L\_CAPTURE.s} and \texttt{MSMON\_MBWU\_L\_CAPTURE.ns} must be separate registers. The Secure instance (\texttt{MSMON\_MBWU\_L\_CAPTURE.s}) accesses the captured long memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (\texttt{MSMON\_MBWU\_L\_CAPTURE.ns}) accesses the captured long memory bandwidth usage monitor used for Non-secure PARTIDs.

\texttt{MSMON\_MBWU\_L\_CAPTURE} can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0890</td>
<td>MSMON_MBWU_CAPTURE_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are \texttt{RW}.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0890</td>
<td>MSMON_MBWU_CAPTURE_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are \texttt{RW}. 

Page 3939
MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_CAPTURE characteristics are:

**Purpose**

MSMON_MBWU_CAPTURE is a 32-bit read-write register that accesses the captured MSMON_MBWU monitor instance selected by MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_s is the Secure memory bandwidth usage monitor capture instance selected by the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_CAPTURE_ns is the Non-secure memory bandwidth usage monitor capture instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_MBWU_CAPTURE is implementation defined.

This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_MBWU_CAPTURE are RES0.

**Attributes**

MSMON_MBWU_CAPTURE is a 32-bit register.

**Field descriptions**

The MSMON_MBWU_CAPTURE bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRDY</td>
<td>Not Ready. The captured NRDY bit from the corresponding instance of MSMON_MBWU. This bit indicates whether the captured monitor value has possibly inaccurate data.</td>
</tr>
<tr>
<td>VALUE</td>
<td>Captured memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.</td>
</tr>
</tbody>
</table>

**NRDY, bit [31]**

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The captured monitor was not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The captured monitor was ready and the VALUE fields is accurate.</td>
</tr>
</tbody>
</table>

**VALUE, bits [30:0]**

Captured memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the captured VALUE field from the corresponding instance of MSMON_MBWU, the count of bytes transferred since the monitor was last reset that meet the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

VALUE captures the MSMON_MBWU.VALUE and preserves any scaling that had been performed on the VALUE field in that register.
Accessing the MSMON_MBWU_CAPTURE

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_MBWU_CAPTURE_s must be accessible from the Secure MPAM feature page. MSMON_MBWU_CAPTURE_ns must be accessible from the Non-secure MPAM feature page.

MSMON_MBWU_CAPTURE_s and MSMON_MBWU_CAPTURE_ns must be separate registers. The Secure instance (MSMON_MBWU_CAPTURE_s) accesses the captured memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_MBWU_CAPTURE_ns) accesses the captured memory bandwidth usage monitor used for Non-secure PARTIDs.

MSMON_MBWU_CAPTURE can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0868</td>
<td>MSMON_MBWU_CAPTURE_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are \textit{RW}.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0868</td>
<td>MSMON_MBWU_CAPTURE_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are \textit{RW}.

27/09/2019 18:23, 6134483bd14dc8c12a99c984cbfe3431cc1c9707
Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
MSMON_MBWU_L, MPAM Long Memory Bandwidth Usage Monitor Register

The MSMON_MBWU_L characteristics are:

**Purpose**

MSMON_MBWU_L is a 64-bit read-write register that accesses the monitor instance selected by 
MSMON_CFG_MON_SEL. MSMON_MBWU_L_s is the Secure memory bandwidth usage monitor instance selected by 
the Secure instance of MSMON_CFG_MON_SEL. MSMON_MBWU_L_ns is the Non-secure memory bandwidth usage 
monitor instance selected by the Non-secure instance of MSMON_CFG_MON_SEL.

**Configuration**

The power domain of MSMON_mBWU_L is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and 
MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L are IMPLEMENTATION 
DEFINED.

**Attributes**

MSMON_MBWU_L is a 64-bit register.

**Field descriptions**

The MSMON_MBWU_L bit assignments are:

When MPAMF_MBWUMON_IDR.LWD == 0:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>NRDY, not ready. Indicates whether the monitor has possibly inaccurate data.</td>
</tr>
<tr>
<td>62:44</td>
<td>The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.</td>
</tr>
<tr>
<td>43:0</td>
<td>VALUE, long (44-bit) memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.</td>
</tr>
</tbody>
</table>

VALUE is the long count of bytes transferred since the monitor was last reset that meet the criteria set in 
MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by 
MSMON_CFG_MON_SEL.
When MPAMF_MBWUMON_IDR.LWD == 1:

| 63  | 62  | 61  | 59  | 58  | 57  | 56  | 55  | 54  | 53  | 52  | 51  | 50  | 49  | 48  | 47  | 46  | 45  | 44  | 43  | 42  | 41  | 40  | 39  | 38  | 37  | 36  | 35  | 34  | 33  | 32  |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| NRDY | VALUE | VALUE |
| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |

NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

<table>
<thead>
<tr>
<th>NRDY</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The monitor is not ready and the contents of the VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage.</td>
</tr>
<tr>
<td>0b1</td>
<td>The monitor is ready and the VALUE fields is accurate.</td>
</tr>
</tbody>
</table>

VALUE, bits [62:0]

Long (63-bit) memory bandwidth usage counter value if NRDY == 0. Invalid if NRDY == 1.

VALUE is the long count of bytes transferred since the monitor was last reset that meet the criteria set in MSMON_CFG_MBWU_FLT and MSMON_CFG_MBWU_CTL for the monitor instance selected by MSMON_CFG_MON_SEL.

Accessing the MSMON_MBWU_L

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_MBWU_L_s must be accessible from the Secure MPAM feature page. MSMON_MBWU_L_ns must be accessible from the Non-secure MPAM feature page.

MSMON_MBWU_L_s and MSMON_MBWU_L_ns must be separate registers. The Secure instance (MSMON_MBWU_L_s) accesses the long memory bandwidth usage monitor used for Secure PARTIDs, and the Non-secure instance (MSMON_MBWU_L_ns) accesses the long memory bandwidth usage monitor used for Non-secure PARTIDs.

**MSMON_MBWU_L can be accessed through the memory-mapped interfaces:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_s</td>
<td>0x0880</td>
<td>MSMON_MBWU_s</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.

<table>
<thead>
<tr>
<th>Component</th>
<th>Frame</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPAM</td>
<td>MPAMF_BASE_ns</td>
<td>0x0880</td>
<td>MSMON_MBWU_ns</td>
</tr>
</tbody>
</table>

Accesses on this interface are **RW**.
OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

**Purpose**

Used to lock or unlock the OS lock.

**Configuration**

External register OSLAR_EL1 bits [31:0] are architecturally mapped to AArch64 System register OSLAR_EL1[31:0].
External register OSLAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register DBGOSLAR[31:0].

OSLAR_EL1 is in the Core power domain.

If ARMv8.2-Debug is not implemented, it is IMPLEMENTATIONDEFINED whether external debug accesses to OSLAR_EL1 are ignored and return an error when AllowExternalDebugAccess() returns FALSE for the access.

If ARMv8.2-Debug is implemented, external debug accesses to OSLAR_EL1 are ignored and return an error when AllowExternalDebugAccess() returns FALSE for the access.

**Attributes**

OSLAR_EL1 is a 32-bit register.

**Field descriptions**

The OSLAR_EL1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | OSLK | Bits [31:1] |

Reserved, RES0.

**OSLK, bit [0]**

On writes to OSLAR_EL1, bit[0] is copied to the OS lock.

Use EDPRSR.OSLK to check the current status of the lock.

**Accessing the OSLAR_EL1**

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

OSLAR_EL1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug</td>
<td>0x300</td>
<td>OSLAR_EL1</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()`, `AllowExternalDebugAccess()` and `SoftwareLockStatus()` accesses to this register are **WI**.
- When `IsCorePowered()`, `!DoubleLockStatus()`, `AllowExternalDebugAccess()` and `!SoftwareLockStatus()` accesses to this register are **WO**.
- When `IsCorePowered()`, `!DoubleLockStatus()`, `!AllowExternalDebugAccess()` and ARMv8.2-Debug is not implemented accesses to this register are **IMPDEF**.
- Otherwise accesses to this register generate an error response.
PMAUTHSTATUS, Performance Monitors Authentication Status register

The PMAUTHSTATUS characteristics are:

**Purpose**

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for Performance Monitors.

**Configuration**

It is IMPLEMENTATION DEFINED whether PMAUTHSTATUS is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is OPTIONAL, and is required for CoreSight compliance. Arm recommends that this register is implemented.

**Attributes**

PMAUTHSTATUS is a 32-bit register.

**Field descriptions**

The PMAUTHSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>SNID</td>
</tr>
<tr>
<td>29</td>
<td>SID</td>
</tr>
<tr>
<td>28</td>
<td>NSNID</td>
</tr>
<tr>
<td>27</td>
<td>NSID</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SNID, bits [7:6]**

Holds the same value as DBGAUTHSTATUS_EL1.SNID.

**SID, bits [5:4]**

Secure invasive debug. Possible values of this field are:

<table>
<thead>
<tr>
<th>SID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NSNID, bits [3:2]**

Holds the same value as DBGAUTHSTATUS_EL1.NSNID.
NSID, bits [1:0]

Non-secure invasive debug. Possible values of this field are:

<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Accessing the PMAUTHSTATUS

PMAUTHSTATUS can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFB8</td>
<td>PMAUTHSTATUS</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
PMCCFILTR_EL0, Performance Monitors Cycle Counter Filter Register

The PMCCFILTR_EL0 characteristics are:

**Purpose**

Determines the modes in which the Cycle Counter, PMCCNTR_EL0, increments.

**Configuration**

External register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCCFILTR_EL0[31:0].

External register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCCFILTR[31:0].

PMCCFILTR_EL0 is in the Core power domain.

On a Warm or Cold reset RW fields in this register reset:

- To architecturally **UNKNOWN** values if the reset is to an Exception level that is using AArch64.
- To 0 if the reset is to an Exception level that is using AArch32.

The register is not affected by an External debug reset.

**Attributes**

PMCCFILTR_EL0 is a 32-bit register.

**Field descriptions**

The PMCCFILTR_EL0 bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| P   | U   | NSK | NS  | M   | RES0 | SH  | RES0 |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**P, bit [31]**

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMCCFILTR_EL0.NSK bit.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count cycles in EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in EL1.</td>
</tr>
</tbody>
</table>

**U, bit [30]**

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMCCFILTR_EL0.NSU bit.

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count cycles in EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count cycles in EL0.</td>
</tr>
</tbody>
</table>
NSK, bit [29]

When HaveEL(EL3):

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Non-secure EL1 are counted. Otherwise, cycles in Non-secure EL1 are not counted.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When HaveEL(EL3):

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.U bit, cycles in Non-secure EL0 are counted. Otherwise, cycles in Non-secure EL0 are not counted.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When HaveEL(EL2):

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If Secure EL2 is implemented, counting in Secure EL2 is further controlled by the PMCCFILTR_EL0.SH bit.

<table>
<thead>
<tr>
<th>NSH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not count cycles in EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count cycles in EL2.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

M, bit [26]

When HaveEL(EL3):

Secure EL3 filtering bit.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Secure EL3 are counted. Otherwise, cycles in Secure EL3 are not counted.

Most applications can ignore this field and set its value to 0.

Note

This field is not visible in the AArch32 PMCCFILTR System register.
Otherwise:

Reserved, RES0.

**Bit [25]**

Reserved, RES0.

**SH, bit [24]**

When ARMv8.4-SecEL2 is implemented:

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMCCFILTR_EL0.NSH bit, cycles in Secure EL2 are counted.

Otherwise, cycles in Secure EL2 are not counted.

If Secure EL2 is not implemented or is disabled, this field is RES0.

---

**Note**

This field is not visible in the AArch32 PMCCFILTR System register.

---

Otherwise:

Reserved, RES0.

**Bits [23:0]**

Reserved, RES0.

**Accessing the PMCCFILTR_EL0**

---

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

---

**PMCCFILTR_EL0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x47C</td>
<td>PMCCFILTR_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are RO.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are RW.
- Otherwise accesses to this register generate an error response.
The PMCCNTR_EL0 characteristics are:

**Purpose**

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile for more information.

PMCCFILTR_EL0 determines the modes and states in which the PMCCNTR_EL0 can increment.

**Configuration**

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register PMCCNTR_EL0[63:0].

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register PMCCNTR[63:0].

PMCCNTR_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMCCNTR_EL0 is a 64-bit register.

**Field descriptions**

The PMCCNTR_EL0 bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CCNT |

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| CCNT |

**CCNT, bits [63:0]**

Cycle count. Depending on the values of PMCR_EL0.\{LC,D\}, the cycle count increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to PMCR_EL0.C sets this field to 0.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMCCNTR_EL0**

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.
**PMCCNTR_EL0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x0F8</td>
<td>PMCCNTR_EL0</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x0FC</td>
<td>PMCCNTR_EL0</td>
<td>63:32</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMCEID0, Performance Monitors Common Event Identification register 0

The PMCEID0 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range `0x0000` to `0x001F`

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

For more information about the common events and the use of the PMCEIDn registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Note**

- Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.
- This view of the register was previously called PMCEID0_EL0.

**Configuration**

External register PMCEID0 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[31:0].

External register PMCEID0 bits [31:0] are architecturally mapped to AArch32 System register PMCEID0[31:0].

PMCEID0 is in the Core power domain.

**Attributes**

PMCEID0 is a 32-bit register.

**Field descriptions**

The PMCEID0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|

**ID<\text{n}>, bit \text{[n]}, for n = 0 to 31**

ID[n] corresponds to common event n.

For each bit:

<table>
<thead>
<tr>
<th>ID&lt;\text{n}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features.
and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

## Accessing the PMCEID0

### Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

### PMCEID0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE20</td>
<td>PMCEID0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
The PMCEID1 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x020 to 0x03F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

For more information about the common events and the use of the PMCEIDn registers see the section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Note**

- Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.
- This view of the register was previously called PMCEID1_EL0.

**Configuration**

External register PMCEID1 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[31:0].

External register PMCEID1 bits [31:0] are architecturally mapped to AArch32 System register PMCEID1[31:0].

PMCEID1 is in the Core power domain.

**Attributes**

PMCEID1 is a 32-bit register.

**Field descriptions**

The PMCEID1 bit assignments are:

<table>
<thead>
<tr>
<th>ID[n]</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features.
and has an event number that can be represented in the PMCEID\(<n>\) registers of that earlier version of the PMU architecture.

## Accessing the PMCEID1

**Note**

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

### PMCEID1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE24</td>
<td>PMCEID1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
PMCEID2, Performance Monitors Common Event Identification register 2

The PMCEID2 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x4000 to 0x401F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

---

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

---

For more information about the common events and the use of the PMCEIDn registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

External register PMCEID2 bits [31:0] are architecturally mapped to AArch64 System register PMCEID0_EL0[63:32].

External register PMCEID2 bits [63:32] are architecturally mapped to AArch32 System register PMCEID2[31:0].

PMCEID2 is in the Core power domain.

This register is present only when ARMv8.1-PMU is implemented. Otherwise, direct accesses to PMCEID2 are RES0.

**Attributes**

PMCEID2 is a 32-bit register.

**Field descriptions**

The PMCEID2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**IDhi<n>**, bit [n], for n = 0 to 31

IDhi[n] corresponds to common event (0x4000 + n).

For each bit:

<table>
<thead>
<tr>
<th><strong>IDhi&lt;n&gt;</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The common event is not implemented, or not counted.</td>
</tr>
<tr>
<td>0b1</td>
<td>The common event is implemented.</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

---

**Note**
Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

## Accessing the PMCEID2

### Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

### PMCEID2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE28</td>
<td>PMCEID2</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
PMCEID3, Performance Monitors Common Event Identification register 3

The PMCEID3 characteristics are:

**Purpose**

Defines which common architectural events and common microarchitectural events are implemented, or counted, using PMU events in the range 0x4020 to 0x403F.

When the value of a bit in the register is 1 the corresponding common event is implemented and counted.

**Note**

Arm recommends that, if a common event is never counted, the value of the corresponding register bit is 0.

For more information about the common events and the use of the PMCEIDn registers see The section describing 'Event numbers and common events' in chapter D5 'The Performance Monitors Extension' of the Arm Architecture Reference Manual, for Armv8-A architecture profile.

**Configuration**

External register PMCEID3 bits [31:0] are architecturally mapped to AArch64 System register PMCEID1_EL0[63:32].

External register PMCEID3 bits [63:32] are architecturally mapped to AArch32 System register PMCEID3[31:0].

PMCEID3 is in the Core power domain.

This register is present only when ARMv8.1-PMU is implemented. Otherwise, direct accesses to PMCEID3 are reserved.

**Attributes**

PMCEID3 is a 32-bit register.

**Field descriptions**

The PMCEID3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td><strong>IDhi&lt;n&gt;</strong>, bit [n], for n = 0 to 31</td>
</tr>
</tbody>
</table>

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional common event.

**Note**
Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID\langle n\rangle registers of that earlier version of the PMU architecture.

**Accessing the PMCEID3**

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**PMCEID3 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE2C</td>
<td>PMCEID3</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
The PMCFGR characteristics are:

**Purpose**

Contains PMU-specific configuration data.

**Configuration**

PMCFGR is in the Core power domain.

**Attributes**

PMCFGR is a 32-bit register.

**Field descriptions**

The PMCFGR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Assignment</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28</td>
<td>NCG, bits [31:28]</td>
</tr>
<tr>
<td>27 26 25 24</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>23 22 21 20</td>
<td>UEN, bit [19]. User-mode Enable Register</td>
</tr>
<tr>
<td>19 18 17 16</td>
<td>PMUSERENR_EL0 is not visible in the external debug interface, so this bit is RAZ.</td>
</tr>
<tr>
<td>15 14 13 12 11</td>
<td>NA, bit [17]. This feature is not supported, so this bit is RAZ.</td>
</tr>
<tr>
<td>10 9 8 7 6 5 4</td>
<td>EX, bit [16]. Export supported. Value is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>3 2 1 0</td>
<td>PMCR_EL0.X is RES0.</td>
</tr>
<tr>
<td></td>
<td>PMCR_EL0.X is read/write.</td>
</tr>
</tbody>
</table>

This feature is not supported, so this field is RAZ.

Reserved, RES0.

User-mode Enable Register supported. PMUSERENR_EL0 is not visible in the external debug interface, so this bit is RAZ.

This feature is not supported, so this bit is RAZ.

This feature is not supported, so this bit is RAZ.
CCD, bit [15]

Cycle counter has prescale. This is RES1 if AArch32 is supported at any EL, and RAZ otherwise.

<table>
<thead>
<tr>
<th>CCD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>PMCR_EL0.D is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>PMCR_EL0.D is read/write.</td>
</tr>
</tbody>
</table>

CC, bit [14]

Dedicated cycle counter (counter 31) supported. This bit is RAO.

SIZE, bits [13:8]

Size of counters, minus one. This field defines the size of the largest counter implemented by the Performance Monitors Unit.

From Armv8, the largest counter is 64-bits, so the value of this field is 0b111111.

This field is used by software to determine the spacing of the counters in the memory-map. From Armv8, the counters are a doubleword-aligned addresses.

N, bits [7:0]

Number of counters implemented in addition to the cycle counter, PMCCNTR_EL0. The maximum number of event counters is 31.

<table>
<thead>
<tr>
<th>N</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Only PMCCNTR_EL0 implemented.</td>
</tr>
<tr>
<td>0x1</td>
<td>PMCCNTR_EL0 plus one event counter implemented.</td>
</tr>
</tbody>
</table>

and so on up to 0b00011111, which indicates PMCCNTR_EL0 and 31 event counters implemented.

Accessing the PMCFGR

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCFGR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE00</td>
<td>PMCFGR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
**PMCID1SR, CONTEXTIDR_EL1 Sample Register**

The PMCID1SR characteristics are:

**Purpose**

Contains the sampled value of CONTEXTIDR_EL1, captured on reading PMPCSR[31:0].

**Configuration**

PMCID1SR is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.2-PCSample is implemented. Otherwise, direct accesses to PMCID1SR are RE0.

**Note**

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of EDDEVID.PCSample.

**Attributes**

PMCID1SR is a 32-bit register.

**Field descriptions**

The PMCID1SR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**CONTEXTIDR_EL1, bits [31:0]**

Context ID. The value of CONTEXTIDR that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample was generated:

- If EL1 is using AArch64, then the Context ID is sampled from CONTEXTIDR_EL1.
- If EL1 is using AArch32, then the Context ID is sampled from CONTEXTIDR. If EL3 is implemented and is using AArch32, then CONTEXTIDR is a banked register and PMCID1SR samples the current banked copy of CONTEXTIDR for the Security state that is associated with the most recent PMPCSR sample.

Because the value written to PMCID1SR is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether PMCID1SR is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMCID1SR**

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
**PMCID1SR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x208</td>
<td>PMCID1SR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()` and `!OSLockStatus()` accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x228</td>
<td>PMCID1SR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()` and `!OSLockStatus()` accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
The PMCID2SR characteristics are:

**Purpose**

Contains the sampled value of CONTEXTIDR_EL2, captured on reading PMPCSR[31:0].

**Configuration**

PMCID2SR is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.2-PCSample is implemented and HaveEL(EL2). Otherwise, direct accesses to PMCID2SR are RES0.

**Note**

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of EDDEVID.PCSample.

**Attributes**

PMCID2SR is a 32-bit register.

**Field descriptions**

The PMCID2SR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CONTEXTIDR_EL2, bits [31:0]

Context ID. The value of CONTEXTIDR_EL2 that is associated with the most recent PMPCSR sample. When the most recent PMPCSR sample was generated:

- If EL2 is using AArch64, then this field is set to the Context ID sampled from CONTEXTIDR_EL2.
- If EL2 is using AArch32, then this field is set to an UNKNOWN value.

Because the value written to PMCID2SR is an indirect read of CONTEXTIDR_EL2, it is CONSTRAINED UNPREDICTABLE whether PMCID2SR is set to the original or new value if PMPCSR samples:

- An instruction that writes to CONTEXTIDR_EL2.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMCID2SR**

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.
PMCID2SR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x22C</td>
<td>PMCID2SR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
The PMCIDR0 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is IMPLEMENTATION DEFINED whether PMCIDR0 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMCIDR0 is a 32-bit register.

**Field descriptions**

The PMCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit 31-8</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-24</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>23-8</td>
<td>Preamble (PRMBL_0)</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_0, bits [7:0]**

Preamble. Must read as 0x0D.

**Accessing the PMCIDR0**

**PMCIDR0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFFF0</td>
<td>PMCIDR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
The PMCIDR1 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is IMPLEMENTATION DEFINED whether PMCIDR1 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMCIDR1 is a 32-bit register.

**Field descriptions**

The PMCIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>CLASS</td>
</tr>
<tr>
<td>29</td>
<td>PRMBL_1</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**CLASS, bits [7:4]**

Component class. Reads as 0x9, debug component.

**PRMBL_1, bits [3:0]**

Preamble. RAZ.

**Accessing the PMCIDR1**

**PMCIDR1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFF4</td>
<td>PMCIDR1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
• Otherwise accesses to this register generate an error response.
PMCIDR2, Performance Monitors Component Identification Register 2

The PMCIDR2 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMCIDR2 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMCIDR2 is a 32-bit register.

**Field descriptions**

The PMCIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>PRMBL_2</td>
</tr>
<tr>
<td></td>
<td>Bit assignments are:</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_2, bits [7:0]**

Preamble. Must read as 0x05.

**Accessing the PMCIDR2**

**PMCIDR2 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFFF8</td>
<td>PMCIDR2</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
The PMCIDR3 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Component identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMCIDR3 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMCIDR3 is a 32-bit register.

**Field descriptions**

The PMCIDR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | PRMBL_3 |

**Bits [31:8]**

Reserved, RES0.

**PRMBL_3, bits [7:0]**

Preamble. Must read as 0x81.

**Accessing the PMCIDR3**

**PMCIDR3 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFFC</td>
<td>PMCIDR3</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
PMCNTENCLR_EL0, Performance Monitors Count Enable Clear register

The PMCNTENCLR_EL0 characteristics are:

**Purpose**

Disables the Cycle Count Register, PMCCNTR_EL0, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

**Configuration**

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCNTENCLR_EL0[31:0].

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENCLR[31:0].

PMCNTENCLR_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMCNTENCLR_EL0 is a 32-bit register.

**Field descriptions**

The PMCNTENCLR_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**C, bit [31]**

PMCCNTR_EL0 disable bit. Disables the cycle counter register. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter is enabled. When written, disables the cycle counter.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter disable bit for PMEVCNTR<n>_EL0.

If PMCFGR.N is less than 31, bits [30:PMCFGR.N] are RAZ/WI.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 is enabled. When written, disables PMEVCNTR&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.
Accessing the PMCNTENCLR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCNTENCLR_EL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xC20</td>
<td>PMCNTENCLR_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMCNTENSET_EL0, Performance Monitors Count Enable Set register

The PMCNTENSET_EL0 characteristics are:

**Purpose**

Enables the Cycle Count Register, PMCCNTR_EL0, and any implemented event counters PMEVCNTR<n>. Reading this register shows which counters are enabled.

**Configuration**

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMCNTENSET_EL0[31:0].

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMCNTENSET[31:0].

PMCNTENSET_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMCNTENSET_EL0 is a 32-bit register.

**Field descriptions**

The PMCNTENSET_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

C, bit [31]

**PMCCNTR_EL0** enable bit. Enables the cycle counter register. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter is enabled. When written, enables the cycle counter.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter enable bit for PMEVCNTR<n>_EL0.

If PMCFGR.N is less than 31, bits [30:PMCFGR.N] are RAZ/WI.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 event counter is enabled. When written, enables PMEVCNTR&lt;n&gt;_EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.
Accessing the PMCNTENSET_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCNTENSET_EL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xC00</td>
<td>PMCNTENSET_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

**Purpose**

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

**Configuration**

External register PMCR_EL0 bits [7:0] are architecturally mapped to AArch32 System register PMCR[7:0].

External register PMCR_EL0 bits [7:0] are architecturally mapped to AArch64 System register PMCR_EL0[7:0].

PMCR_EL0 is in the Core power domain. Some or all RW fields of this register have defined reset values. The field descriptions identify when the reset values apply.

This register is only partially mapped to the internal PMCR System register. An external agent must use other means to discover the information held in PMCR[31:11], such as accessing PMCFGR and the ID registers.

**Attributes**

PMCR_EL0 is a 32-bit register.

**Field descriptions**

The PMCR_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RAZ/WI |    | RES0 | LP | LC | DP | X | D | C | P | E |

**Bits [31:11]**

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

**Bits [10:8]**

Reserved, RES0.

**LP, bit [7]**

When ARMv8.5-PMU is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by a counter overflow bit.

<table>
<thead>
<tr>
<th>LP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR&lt;n&gt;_EL0[31:0].</td>
</tr>
<tr>
<td>0b1</td>
<td>Event counter overflow on increment that causes unsigned overflow of PMEVCNTR&lt;n&gt;_EL0[63:0].</td>
</tr>
</tbody>
</table>

If EL2 is implemented and MDCR_EL2.HPMN or HDCR.HPMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [HDCR.HPMN..(PMCR_EL0.N-1)] or [MDCR_EL2.HPMN:(PMCR_EL0.N-1)].
The effect of `MDCR_EL2.HPMN` or `HDCR.HPMN` on the operation of this bit always applies if EL2 is implemented, at all Exception levels including EL2 and EL3, and regardless of whether EL2 is enabled in the current Security state. For more information, see the description of `MDCR_EL2.HPMN` or `HDCR.HPMN`.

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is RW or RAZ/WI.

**Otherwise:**

Reserved, RES0.

**LC, bit [6]**

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

<table>
<thead>
<tr>
<th>LC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counter overflow on increment that causes unsigned overflow of <code>PMCCNTR_EL0[31:0]</code>.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counter overflow on increment that causes unsigned overflow of <code>PMCCNTR_EL0[63:0]</code>.</td>
</tr>
</tbody>
</table>

Arm deprecates use of `PMCR_EL0.LC = 0`.

In an AArch64 only implementation, this field is RES1.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**DP, bit [5]**

*When HaveEL(EL3) or (ARMv8.1-PMU is implemented and HaveEL(EL2)):*

Disable cycle counter when event counting is prohibited. The possible values of this bit are:

<table>
<thead>
<tr>
<th>DP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting by <code>PMCCNTR_EL0</code> is not affected by this bit.</td>
</tr>
<tr>
<td>0b1</td>
<td>When event counting for counters in the range <code>[0..(MDCR_EL2.HPMN-1)]</code> is prohibited, cycle counting by <code>PMCCNTR_EL0</code> is disabled.</td>
</tr>
</tbody>
</table>

For more information about the interaction between the Performance Monitors and EL3, see ‘Effect of EL3 and EL2’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally UNKNOWN if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

**Otherwise:**

Reserved, RES0.

**X, bit [4]**

*When the implementation includes a PMU event export bus:*

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.
### PMCR_EL0, Performance Monitors Control Register

<table>
<thead>
<tr>
<th>X</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not export events.</td>
</tr>
<tr>
<td>0b1</td>
<td>Export events where not prohibited.</td>
</tr>
</tbody>
</table>

This field enables the exporting of events over an **IMPLEMENTATION DEFINED** PMU event export bus to another device, for example to an **OPTIONAL** PE trace unit.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally **UNKNOWN** if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

**Otherwise:**

Reserved, RAZ/WI.

### D, bit [3]

**When AArch32 is supported at any Exception level:**

Clock divider.

<table>
<thead>
<tr>
<th>D</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When enabled, ( \text{PMCCNTR_EL0} ) counts every clock cycle.</td>
</tr>
<tr>
<td>0b1</td>
<td>When enabled, ( \text{PMCCNTR_EL0} ) counts once every 64 clock cycles.</td>
</tr>
</tbody>
</table>

If PMCR_EL0.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally **UNKNOWN** if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

**Otherwise:**

Reserved, RES0.

### C, bit [2]

Cycle counter reset. The effects of writing to this bit are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset ( \text{PMCCNTR_EL0} ) to zero.</td>
</tr>
</tbody>
</table>

This bit is always RAZ.

**Note**

Resetting \( \text{PMCCNTR_EL0} \) does not change the cycle counter overflow bit.

Access to this field is **WO**.

### P, bit [1]

Event counter reset. The effects of writing to this bit are:
Meaning

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action.</td>
</tr>
<tr>
<td>0b1</td>
<td>Reset all event counters, not including PMCCNTR_EL0 to zero.</td>
</tr>
</tbody>
</table>

This bit is always RAZ.

Note

Resetting the event counters does not change the event counter overflow bits.

If ARMv8.5-PMU is implemented, the value of MDAR_EL2.HLP, or PMCR_EL0.LP is ignored and bits [63:0] of all event counters are reset.

Access to this field is WO.

E, bit [0]

Enable.

<table>
<thead>
<tr>
<th>E</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>All event counters in the range [0..(PMN-1)] and PMCCNTR_EL0 are disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>All event counters in the range [0..(PMN-1)] and PMCCNTR_EL0 are enabled by PMCNTENSET_EL0.</td>
</tr>
</tbody>
</table>

If EL2 is implemented then:

- If EL2 is using AArch32, PMN is HDCR.HPMN.
- If EL2 is using AArch64, PMN is MDAR_EL2.HPMN.
- If PMN is less than PMCR_EL0.N, this bit does not affect the operation of event counters in the range [PMN..(PMCR_EL0.N-1)].

If EL2 is not implemented, PMN is PMCR_EL0.N.

Note

The effect of MDAR_EL2.HPMN or HDCR.HPMN on the operation of this bit applies if EL2 is implemented regardless of whether EL2 is enabled in the current Security state. For more information, see the description of MDAR_EL2.HPMN or HDCR.HPMN.

On a Warm reset, this field resets to 0.

Accessing the PMCR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCR_EL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE04</td>
<td>PMCR_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are RO.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are RW.
• Otherwise accesses to this register generate an error response.
PMDEVAFF0, Performance Monitors Device Affinity register 0

The PMDEVAFF0 characteristics are:

**Purpose**

Copy of the low half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMDEVAFF0 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required if the external interface to the PMU is implemented.

**Attributes**

PMDEVAFF0 is a 32-bit register.

**Field descriptions**

The PMDEVAFF0 bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|        |        | MPIDR_EL1 low half |

**MPIDR_EL1 low half, bits [31:0]**

MPIDR_EL1 low half. Read-only copy of the low half of MPIDR_EL1, as seen from the highest implemented Exception level.

**Accessing the PMDEVAFF0**

PMDEVAFF0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFA8</td>
<td>PMDEVAFF0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
PMDEVAFF1, Performance Monitors Device Affinity register 1

The PMDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE MPIDR_EL1 register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

Configuration

It is IMPLEMENTATION DEFINED whether PMDEVAFF1 is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required if the external interface to the PMU is implemented.

Attributes

PMDEVAFF1 is a 32-bit register.

Field descriptions

The PMDEVAFF1 bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     | MPIDR_EL1 high half |

MPIDR_EL1 high half, bits [31:0]

MPIDR_EL1 high half. Read-only copy of the high half of MPIDR_EL1, as seen from the highest implemented Exception level.

Accessing the PMDEVAFF1

PMDEVAFF1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFAC</td>
<td>PMDEVAFF1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
PMDEVARCH, Performance Monitors Device Architecture register

The PMDEVARCH characteristics are:

**Purpose**

Identifies the programmers' model architecture of the Performance Monitor component.

**Configuration**

It is implementation defined whether PMDEVARCH is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

PMDEVARCH is a 32-bit register.

**Field descriptions**

The PMDEVARCH bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ARCHITECT</td>
<td>Defines the architecture of the component. For Performance Monitors, this is Arm Limited.</td>
</tr>
<tr>
<td>30</td>
<td>PRESENT</td>
<td>When set to 1, indicates that the DEVARCH is present.</td>
</tr>
<tr>
<td>29</td>
<td>REVISION</td>
<td>Defines the architecture revision. For architectures defined by Arm this is the minor revision. For Performance Monitors, the revision defined by Armv8 is 0x0.</td>
</tr>
<tr>
<td>28</td>
<td>ARCHID</td>
<td>Defines this part to be an Armv8 debug component. For architectures defined by Arm this is further subdivided. For Performance Monitors:</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>Bits [15:12] are the architecture version, 0x2.</td>
</tr>
</tbody>
</table>

ARCHITECT, bits [31:21]

Defines the architecture of the component. For Performance Monitors, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

PRESENT, bit [20]

When set to 1, indicates that the DEVARCH is present.

This field is 1 in Armv8.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

For Performance Monitors, the revision defined by Armv8 is 0x0.

All other values are reserved.

ARCHID, bits [15:0]

Defines this part to be an Armv8 debug component. For architectures defined by Arm this is further subdivided.

For Performance Monitors:

- Bits [15:12] are the architecture version, 0x2.
• Bits [11:0] are the architecture part number, 0xA16.

This corresponds to Performance Monitors architecture version PMUv3.

---

**Note**

The PMUv3 memory-mapped programmers' model can be used by devices other than Armv8 processors. Software must determine whether the PMU is attached to an Armv8 processor by using the `PMDEVAFF0` and `PMDEVAFF1` registers to discover the affinity of the PMU to any Armv8 processors.

---

### Accessing the PMDEVARCH

**PMDEVARCH can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFBC</td>
<td>PMDEVARCH</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or `IsCorePowered()` accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
PMDEVID, Performance Monitors Device ID register

The PMDEVID characteristics are:

**Purpose**

Provides information about features of the Performance Monitors implementation.

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMDEVID is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required from Armv8.2 and in any implementation that includes ARMv8.2-PCSample. Otherwise, its location is **RES0**.

**Note**

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of **EDDEVID**.PCSample.

**Attributes**

PMDEVID is a 32-bit register.

**Field descriptions**

The PMDEVID bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td><strong>PCSample</strong></td>
</tr>
</tbody>
</table>

**Bits [31:4]**

Reserved, **RES0**.

**PCSample, bits [3:0]**

Indicates the level of PC Sample-based Profiling support using Performance Monitors registers. Permitted values of this field are:

<table>
<thead>
<tr>
<th><strong>PCSample</strong></th>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>PC Sample-based Profiling Extension is not implemented in the Performance Monitors register space.</td>
</tr>
<tr>
<td>0b0001</td>
<td>PC Sample-based Profiling Extension is implemented in the Performance Monitors register space.</td>
</tr>
</tbody>
</table>

All other values are reserved.

ARMv8.2-PCSample implements the functionality identified by the value 0b0001.
Accessing the PMDEVID

PMDEVID can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFC8</td>
<td>PMDEVID</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
PMDEVTYPE, Performance Monitors Device Type register

The PMDEVTYPE characteristics are:

**Purpose**

Indicates to a debugger that this component is part of a PEs performance monitor interface.

**Configuration**

It is `IMPLEMENTATION DEFINED` whether PMDEVTYPE is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is `OPTIONAL`.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

**Attributes**

PMDEVTYPE is a 32-bit register.

**Field descriptions**

The PMDEVTYPE bit assignments are:

<table>
<thead>
<tr>
<th>Bit Width</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Reserved</td>
<td>RES0</td>
</tr>
<tr>
<td>4</td>
<td>Subtype</td>
<td>SUB</td>
</tr>
<tr>
<td>3</td>
<td>Major Type</td>
<td>MAJOR</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SUB, bits [7:4]**

Subtype. Must read as 0x1 to indicate this is a component within a PE.

**MAJOR, bits [3:0]**

Major type. Must read as 0x6 to indicate this is a performance monitor component.

**Accessing the PMDEVTYPE**

**PMDEVTYPE can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFCC</td>
<td>PMDEVTYPE</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
PMEVCNTR\textless n\textgreater _EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR\textless n\textgreater _EL0 characteristics are:

**Purpose**

Holds event counter n, which counts events, where n is 0 to 30.

**Configuration**

External register PMEVCNTR\textless n\textgreater _EL0 bits [31:0] are architecturally mapped to AArch64 System register PMEVCNTR\textless n\textgreater _EL0[31:0].

External register PMEVCNTR\textless n\textgreater _EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVCNTR\textless n\textgreater [31:0].

PMEVCNTR\textless n\textgreater _EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. The field descriptions identify when the reset values apply.

**Attributes**

PMEVCNTR\textless n\textgreater _EL0 is a 64-bit register.

**Field descriptions**

The PMEVCNTR\textless n\textgreater _EL0 bit assignments are:

**When ARMv8.5-PMU is implemented:**

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [63:0]**

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If the highest implemented Exception level is using AArch32, the optional external interface to the performance monitors is implemented, and the PMCR.LP and HDCR.HLP bits are RAZ/WI, then locations in the external interface to the performance monitors that map to PMEVCNTR\textless n\textgreater _EL0[63:32] return UNKNOWN values on reads.

If the implementation does not support AArch64 at any Exception level, bits [63:32] of the event counters are not required to be implemented.

This field resets to an architecturally UNKNOWN value.

**Otherwise:**

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**Bits [31:0]**

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.
This field resets to an architecturally **UNKNOWN** value.

**Accessing the PMEVCNTR<\text{n}>_EL0**

External accesses to the performance monitors ignore **PMUSERENR_EL0** and, if implemented, **MDCR_EL2**, {TPM, TPMCR, HPMN} and **MDCR_EL3**. This means that all counters are accessible regardless of the current Exception level or privilege of the access.

If ARMv8.5-PMU is not implemented, when IsCorePowered(), DoubleLockStatus(), OSLockStatus() or !AllowExternalPMUAccess(), 32-bit accesses to **0x004+8×n** have a **CONSTRAINED UNPREDICTABLE** behavior.

---

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

---

**PMEVCNTR<\text{n}>_EL0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x000 + 8n</td>
<td>PMEVCNTR&lt;\text{n}&gt;_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n>_EL0 characteristics are:

**Purpose**

Configures event counter n, where n is 0 to 30.

**Configuration**

External register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMEVTYPER<n>_EL0[31:0].

External register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMEVTYPER<n>[31:0].

PMEVTYPER<n>_EL0 is in the Core power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

If event counter n is not implemented then accesses to this register are:

- **RES0** when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalPMUAccess().
- A **CONSTRAINED UNPREDICTABLE** choice of **RES0** or ERROR otherwise.

**Attributes**

PMEVTYPER<n>_EL0 is a 32-bit register.

**Field descriptions**

The PMEVTYPER<n>_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| P  | U  | NSK | NSU | NSH | M  | MT  | SH  | T  | RES0 | evtCount[15:10] | evtCount[9:0] |

**P, bit [31]**

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMEVTYPER<n>_EL0.NSK bit.

<table>
<thead>
<tr>
<th>P</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events in EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in EL1.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**U, bit [30]**

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMEVTYPER<n>_EL0.NSU bit.

<table>
<thead>
<tr>
<th>U</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events in EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in EL0.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.
NSK, bit [29]

When HaveEL(EL3):

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in Non-secure EL1 are counted.
Otherwise, events in Non-secure EL1 are not counted.

On a Warm reset, this field resets to an architecturally \textit{UNKNOWN} value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When HaveEL(EL3):

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.U bit, events in Non-secure EL0 are counted.
Otherwise, events in Non-secure EL0 are not counted.

On a Warm reset, this field resets to an architecturally \textit{UNKNOWN} value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When HaveEL(EL2):

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If Secure EL2 is implemented, counting in Secure EL2 is further controlled by the PMEVTYPER<n>_EL0.SH bit.

<table>
<thead>
<tr>
<th>NSH</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not count events in EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count events in EL2.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally \textit{UNKNOWN} value.

Otherwise:

Reserved, RES0.

M, bit [26]

When HaveEL(EL3):

Secure EL3 filtering bit.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in Secure EL3 are counted.
Otherwise, events in Secure EL3 are not counted.

Most applications can ignore this field and set its value to 0b0.

---

\textbf{Note}
This field is not visible in the AArch32 PMEVTYPER<n> System register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MT, bit [25]

When ARMv8.6-MTPMU is implemented or IMPLEMENTATION DEFINED multi-threaded PMU Extension is implemented:

Multithreading.

<table>
<thead>
<tr>
<th>MT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Count events only on controlling PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>Count events from any PE with the same affinity at level 1 and above as this PE.</td>
</tr>
</tbody>
</table>

Note

- When the lowest level of affinity consists of logical PEs that are implemented using a multi-threading type approach, an implementation is described as multi-threaded. That is, the performance of PEs at the lowest affinity level is highly interdependent.
- Events from a different thread of a multithreaded implementation are not Attributable to the thread counting the event.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bit [24]

When ARMv8.4-SecEL2 is implemented:

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMEVTYPER<n>_EL0.NSH bit, events in Secure EL2 are counted.

Otherwise, events in Secure EL2 are not counted.

Note

This field is not visible in the AArch32 PMEVTYPER<n> System register.

On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T, bit [23]

When TME is implemented:

Transactional state filtering bit. Controls counting in Transactional state. The possible values of this bit are:
<table>
<thead>
<tr>
<th>T</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This bit has no effect on the filtering of events.</td>
</tr>
<tr>
<td>0b1</td>
<td>Do not count events in Transactional state.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**Bits [22:16]**

Reserved, RES0.

**evtCount[15:10], bits [15:10]**

*When ARMv8.1-PMU is implemented:*

Extension to evtCount[9:0]. See evtCount[9:0] for more details.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, RES0.

**evtCount[9:0], bits [9:0]**

Event to count. The event number of the event that is counted by event counter **PMEVCNTR<n>_EL0**.

Software must program this field with an event that is supported by the PE being programmed.

The ranges of event numbers allocated to each type of event are shown in Allocation of the PMU event number space.

If evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range **0x0000** to **0x003F**, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- If 16-bit evtCount is implemented, for the range **0x4000** to **0x403F**, no events are counted, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For **IMPLEMENTATION DEFINED** events, it is **UNPREDICTABLE** what event, if any, is counted, and the value returned by a direct or external read of the evtCount field is **UNKNOWN**.

**Note**

**UNPREDICTABLE** means the event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include an event from a set of common **IMPLEMENTATION DEFINED** events, then no event is counted and the value read back on evtCount is the value written.

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMEVTYPEPER<n>_EL0**

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.
PMEVTPRER<n>_EL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x400 + 4n</td>
<td>PMEVTYPER&lt;n&gt;_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR_EL1 characteristics are:

**Purpose**

Disables the generation of interrupt requests on overflows from the Cycle Count Register, PMCCNTR_EL0, and the event counters PMEVCNTR<n>_EL0. Reading the register shows which overflow interrupt requests are enabled.

**Configuration**

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register PMINTENCLR_EL1[31:0].

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register PMINTENCLR[31:0].

PMINTENCLR_EL1 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMINTENCLR_EL1 is a 32-bit register.

**Field descriptions**

The PMINTENCLR_EL1 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  | P<n>, bit [n] |

**C, bit [31]**

PMCCNTR_EL0 overflow interrupt request disable bit. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow interrupt request disable bit for PMEVCNTR<n>_EL0.

If PMCFGR.N is less than 31, bits [30:PMCFGR.N] are RAZ/WI.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that the PMEVCNTR&lt;n&gt;_EL0 event counter interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that the PMEVCNTR&lt;n&gt;_EL0 event counter interrupt request is enabled. When written, disables the PMEVCNTR&lt;n&gt;_EL0 interrupt request.</td>
</tr>
</tbody>
</table>
On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

## Accessing the PMINTENCLR_EL1

### Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMINTENCLR_EL1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xC60</td>
<td>PMINTENCLR_EL1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMINTENSET_EL1, Performance Monitors Interrupt Enable Set register

The PMINTENSET_EL1 characteristics are:

**Purpose**

Enables the generation of interrupt requests on overflows from the Cycle Count Register, `PMCCNTR_EL0`, and the event counters `PMEVCNTR<n>_EL0`. Reading the register shows which overflow interrupt requests are enabled.

**Configuration**

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register `PMINTENSET_EL1[31:0]`.

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register `PMINTENSET[31:0]`.

PMINTENSET_EL1 is in the Core power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMINTENSET_EL1 is a 32-bit register.

**Field descriptions**

The PMINTENSET_EL1 bit assignments are:

```
+-----------------+---+
<table>
<thead>
<tr>
<th>C</th>
<th>P&lt;n&gt;, bit [n]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td></td>
</tr>
<tr>
<td>0b1</td>
<td></td>
</tr>
</tbody>
</table>
```

C, bit [31]

`PMCCNTR_EL0` overflow interrupt request enable bit. Possible values are:

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter overflow interrupt request is</td>
</tr>
<tr>
<td></td>
<td>disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter overflow interrupt request is</td>
</tr>
<tr>
<td></td>
<td>enabled. When written, enables the cycle count overflow interrupt</td>
</tr>
<tr>
<td></td>
<td>request.</td>
</tr>
</tbody>
</table>

On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

P<n>, bit [n], for n = 0 to 30

Event counter overflow interrupt request enable bit for `PMEVCNTR<n>_EL0`.

If `PMCFGFR.N` is less than 31, bits [30:`PMCFGFR.N`] are RAZ/WI.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that the <code>PMEVCNTR&lt;n&gt;_EL0</code> event counter</td>
</tr>
<tr>
<td></td>
<td>interrupt request is disabled. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that the <code>PMEVCNTR&lt;n&gt;_EL0</code> event counter</td>
</tr>
<tr>
<td></td>
<td>interrupt request is enabled. When written, enables the <code>PMEVCNTR&lt;n&gt;_EL0</code></td>
</tr>
<tr>
<td></td>
<td>interrupt request.</td>
</tr>
</tbody>
</table>
On a Warm reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the PMINTENSET_EL1**

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**PMINTENSET_EL1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xC40</td>
<td>PMINTENSET_EL1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMITCTRL, Performance Monitors Integration mode Control register

The PMITCTRL characteristics are:

**Purpose**

Enables the Performance Monitors to switch from default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

**Configuration**

It is implementation defined whether PMITCTRL is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

Implementation of this register is optional.

**Attributes**

PMITCTRL is a 32-bit register.

**Field descriptions**

The PMITCTRL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>IME</td>
</tr>
</tbody>
</table>

**Bits [31:1]**

Reserved, RES0.

**IME, bit [0]**

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection. The integration mode behavior is implementation defined.

<table>
<thead>
<tr>
<th>IME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Normal operation.</td>
</tr>
<tr>
<td>0b1</td>
<td>Integration mode enabled.</td>
</tr>
</tbody>
</table>

On a Implementation reset, this field resets to 0.

**Accessing the PMITCTRL**

PMITCTRL can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xF00</td>
<td>PMITCTRL</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.
PMLAR, Performance Monitors Lock Access Register

The PMLAR characteristics are:

**Purpose**

Allows or disallows access to the Performance Monitors registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Performance Monitors registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Performance Monitors registers. It does not, and cannot, prevent all accidental or malicious damage.

**Configuration**

It is IMPLEMENTATION DEFINED whether PMLAR is implemented in the Core power domain or in the Debug power domain.

If ARMv8.3-DoPD is implemented, then ARMv8.0-SoftwareLock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If ARMv8.3-DoPD is not implemented, then this register is in the Debug power domain.

Software uses PMLAR to set or clear the lock, and PMLSR to check the current status of the lock.

**Attributes**

PMLAR is a 32-bit register.

**Field descriptions**

The PMLAR bit assignments are:

**When the Software Lock is implemented:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**KEY, bits [31:0]**

Lock Access control. Writing the key value 0xC5ACCE55 to this field unlocks the lock, enabling write accesses to this component’s registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component’s registers through a memory-mapped interface.

**Otherwise:**

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Otherwise**

**Bits [31:0]**

Reserved, RES0.
Accessing the PMLAR

PMLAR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFB0</td>
<td>PMLAR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.
PMLSR, Performance Monitors Lock Status Register

The PMLSR characteristics are:

**Purpose**

Indicates the current status of the software lock for Performance Monitors registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Performance Monitors registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Performance Monitors registers. It does not, and cannot, prevent all accidental or malicious damage.

**Configuration**

It is IMPLEMENTATION DEFINED whether PMLSR is implemented in the Core power domain or in the Debug power domain. Some or all RW fields of this register have defined reset values, and:

- The register is not affected by a Warm reset.
- If the register is implemented in the Core power domain the reset values apply on a Cold reset, and the register is not affected by an External debug reset.
- If the register is implemented in the Debug power domain the reset values apply on an External debug reset, and the register is not affected by a Cold reset.

If ARMv8.3-DoPD is implemented, then ARMv8.0-SoftwareLock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If ARMv8.3-DoPD is not implemented, then this register is in the Debug power domain.

Software uses PMLAR to set or clear the lock, and PMLSR to check the current status of the lock.

**Attributes**

PMLSR is a 32-bit register.

**Field descriptions**

The PMLSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>nTT</td>
</tr>
<tr>
<td>29</td>
<td>SLK</td>
</tr>
<tr>
<td>28</td>
<td>SLI</td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:3]**

Reserved, RES0.

**nTT, bit [2]**

Not thirty-two bit access required. RAZ.

**SLK, bit [1]**

When the Software Lock is implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when the Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when the software lock is implemented, possible values of this field are:
SLK, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ. For memory-mapped accesses, the value of this field is IMPLEMENTATION DEFINED. Permitted values are:

<table>
<thead>
<tr>
<th>SLI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Software Lock not implemented or not memory-mapped access.</td>
</tr>
<tr>
<td>0b1</td>
<td>Software Lock implemented and memory-mapped access.</td>
</tr>
</tbody>
</table>

Accessing the PMLSR

PMLSR can be accessed through the memory-mapped interfaces:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFB4</td>
<td>PMLSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
PMMIR, Performance Monitors Machine Identification Register

The PMMIR characteristics are:

**Purpose**

Describes Performance Monitors parameters specific to the implementation.

**Configuration**

PMMIR is in the Core power domain.

This register is present only when ARMv8.4-PMU is implemented. Otherwise, direct accesses to PMMIR are RES0.

**Attributes**

PMMIR is a 32-bit register.

**Field descriptions**

The PMMIR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | SLOTS |

**Bits [31:8]**

Reserved, RES0.

**SLOTS, bits [7:0]**

Operation width. The largest value by which the STALL_SLOT event might increment by in a single cycle. If the STALL_SLOT event is implemented, this field must not be zero.

**Accessing the PMMIR**

If the Core power domain is off or in a low-power state, access on this interface returns an Error.

**PMMIR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xE40</td>
<td>PMMIR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsCorePowered(), or DoubleLockStatus(), or OSLockStatus() or !AllowExternalPMUAccess() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
**PMOVSCLR_EL0, Performance Monitors Overflow Flag Status Clear register**

The PMOVSCLR_EL0 characteristics are:

**Purpose**

Contains the state of the overflow bit for the Cycle Count Register, PMCCNTR_EL0, and each of the implemented event counters PMEVCNTR<

Writing to this register clears these bits.

**Configuration**

External register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMOVSCLR_EL0[31:0].

External register PMOVSCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSBR[31:0].

PMOVSCLR_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMOVSCLR_EL0 is a 32-bit register.

**Field descriptions**

The PMOVSCLR_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | P<n>, bit [n] |

**C, bit [31]**

Cycle counter overflow clear bit.

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter has overflowed since this bit was last cleared. When written, clears the cycle counter overflow bit to 0.</td>
</tr>
</tbody>
</table>

PMCCNTR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**P<n>, bit [n], for n = 0 to 30**

Event counter overflow clear bit for PMEVCNTR<n>_EL0.

If PMCFGR.N is less than 31, bits [30:PMCFGR.N] are RAZ/WI.
<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 has overflowed since this bit was last cleared. When written, clears the PMEVCNTR&lt;n&gt;_EL0 overflow bit to 0.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<n>_EL0[31:0] or unsigned overflow of PMEVCNTR<n>_EL0[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.

### Accessing the PMOVSCRL_EL0

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMOVSCRL_EL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xC80</td>
<td>PMOVSCRL_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.
PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set register

The PMOVSSET_EL0 characteristics are:

**Purpose**

Sets the state of the overflow bit for the Cycle Count Register, PMCCNTR_EL0, and each of the implemented event counters PMEVCNTR<n>.

**Configuration**

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMOVSSET_EL0[31:0].

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMOVSSET[31:0].

PMOVSSET_EL0 is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply on a Warm or Cold reset. The register is not affected by an External debug reset.

**Attributes**

PMOVSSET_EL0 is a 32-bit register.

**Field descriptions**

The PMOVSSET_EL0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| C  | P<n>, bit [n] |

C, bit [31]

Cycle counter overflow set bit.

<table>
<thead>
<tr>
<th>C</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means the cycle counter has overflowed since this bit was last cleared. When written, sets the cycle counter overflow bit to 1.</td>
</tr>
</tbody>
</table>

PMCR_EL0.LC controls whether an overflow is detected from unsigned overflow of PMCCNTR_EL0[31:0] or unsigned overflow of PMCCNTR_EL0[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 0 to 30

Event counter overflow set bit for PMEVCNTR<n>_EL0.

If PMCFGR.N is less than 31, bits [30:PMCFGR.N] are RAZ/WI.
PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set register

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 has not overflowed since this bit was last cleared. When written, has no effect.</td>
</tr>
<tr>
<td>0b1</td>
<td>When read, means that PMEVCNTR&lt;n&gt;_EL0 has overflowed since this bit was last cleared. When written, sets the PMEVCNTR&lt;n&gt;_EL0 overflow bit to 1.</td>
</tr>
</tbody>
</table>

If ARMv8.5-PMU is implemented, MDCR_EL2.HLP and PMCR_EL0.LP control whether an overflow is detected from unsigned overflow of PMEVCNTR<n>_EL0[31:0] or unsigned overflow of PMEVCNTR<n>_EL0[63:0].

On a Warm reset, this field resets to an architecturally UNKNOWN value.

**Accessing the PMOVSSET_EL0**

**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**PMOVSSET_EL0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xCC0</td>
<td>PMOVSSET_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
The PMPCSR characteristics are:

**Purpose**

Holds a sampled instruction address value.

**Configuration**

PMPCSR is in the Core power domain. RW fields in this register reset to architecturally **UNKNOWN** values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.2-PCSample is implemented. Otherwise, direct accesses to PMPCSR are **RES0**.

**Note**

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of **EDDEVID.PCSample**.

Support for 64-bit atomic reads is IMPLEMENTATION DEFINED. If 64-bit atomic reads are implemented, a 64-bit read of PMPCSR has the same side-effect as a 32-bit read of PMCSR[31:0] followed by a 32-bit read of PMCSR[63:32], returning the combined value. For example, if the PE is in Debug state then a 64-bit atomic read returns bits[31:0] == 0xFFFFFFFF and bits[63:32] **UNKNOWN**.

**Attributes**

PMPCSR is a 64-bit register.

**Field descriptions**

The PMPCSR bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| **NS** | **EL** | **T** | **RES0** | PC Sample[55:32] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PC Sample[31:0] |

**NS, bit [63]**

Non-secure state sample. Indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

<table>
<thead>
<tr>
<th><strong>NS</strong></th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample is from Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample is from Non-secure state.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**EL, bits [62:61]**

Exception level status sample. Indicates the Exception level that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.
<table>
<thead>
<tr>
<th>EL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Sample is from EL0.</td>
</tr>
<tr>
<td>0b01</td>
<td>Sample is from EL1.</td>
</tr>
<tr>
<td>0b10</td>
<td>Sample is from EL2.</td>
</tr>
<tr>
<td>0b11</td>
<td>Sample is from EL3.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**T, bit [60]**

When TME is implemented:

Transactional state of the sample. Indicates the Transactional state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

<table>
<thead>
<tr>
<th>T</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Sample is from Non-transactional state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Sample is from Transactional state.</td>
</tr>
</tbody>
</table>

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

Otherwise:

Reserved, RES0.

**Bits [59:56]**

Reserved, RES0.

**PC Sample[55:32], bits [55:32]**

Bits[55:32] of the sampled instruction address value. The translation regime that PMPCSR samples can be determined from PMPCSR.(NS,EL).

On a Cold reset, this field resets to an architecturally **UNKNOWN** value.

**PC Sample[31:0], bits [31:0]**

Bits[31:0] of the sampled instruction address value.

PMPCSR[31:0] reads as **0xffffffff** when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If an instruction has retired since the PE left Reset state, then the first read of PMPCSR[31:0] is permitted but not required to return **0xffffffff**.

PMPCSR[31:0] reads as an **UNKNOWN** value when any of the following are true:

- The PE is in Reset state.
- No instruction has retired since the PE left Reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No instruction has retired since the last read of PMPCSR[31:0].

For the cases where a read of PMPCSR[31:0] returns **0xffffffff** or an **UNKNOWN** value, the read has the side-effect of setting PMPCSR[63:32], **PMCID1SR**, **PMCID2SR**, and **PMVIDSR** to **UNKNOWN** values.

Otherwise, a read of PMPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to PMPCSR[63:32], **PMCID1SR**, **PMCID2SR**, and **PMVIDSR**. The translation regime that PMPCSR samples can be determined from PMPCSR.(NS,EL).

For a read of PMPCSR[31:0] from the memory-mapped interface, if PMLSR.SLK == 1, meaning the **OPTIONAL** Software Lock is locked, then the side-effect of the access does not occur and PMPCSR[63:32], **PMCID1SR**, **PMCID2SR**, and **PMVIDSR** are unchanged.
On a Cold reset, this field resets to an architecturally UNKNOWN value.

## Accessing the PMPCSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile

**PMPCSR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x200</td>
<td>PMPCSR</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x204</td>
<td>PMPCSR</td>
<td>63:32</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x220</td>
<td>PMPCSR</td>
<td>31:0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x224</td>
<td>PMPCSR</td>
<td>63:32</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
The PMPIDR0 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMPIDR0 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMPIDR0 is a 32-bit register.

**Field descriptions**

The PMPIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>29</td>
<td>PART_0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PART_0, bits [7:0]**

Part number, least significant byte.

**Accessing the PMPIDR0**

PMPIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFE0</td>
<td>PMPIDR0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
The PMPIDR1 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMPIDR1 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMPIDR1 is a 32-bit register.

**Field descriptions**

The PMPIDR1 bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9  | Bit 8  | Bit 7  | Bit 6  | Bit 5  | Bit 4  | Bit 3  | Bit 2  | Bit 1  | Bit 0  |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| RES0  |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|       | DES_0 |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |
|       |       | PART_1|       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |

**Bits [31:8]**

Reserved, RES0.

**DES_0, bits [7:4]**

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is 0b1011.

**PART_1, bits [3:0]**

Part number, most significant nibble.

**Accessing the PMPIDR1**

PMPIDR1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFE4</td>
<td>PMPIDR1</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
• Otherwise accesses to this register generate an error response.
The PMPIDR2 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION** **DEFINED** whether PMPIDR2 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMPIDR2 is a 32-bit register.

**Field descriptions**

The PMPIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit Index</th>
<th>Bit Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-8</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7-4</td>
<td>REVISION</td>
<td>Part major revision. Parts can also use this field to extend Part number to 16-bits.</td>
</tr>
<tr>
<td>3</td>
<td>JEDEC</td>
<td>RAO. Indicates a JEP106 identity code is used.</td>
</tr>
<tr>
<td>2-0</td>
<td>DES_1</td>
<td>Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is 0b011.</td>
</tr>
</tbody>
</table>
Accessing the PMPIDR2

PMPIDR2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFE8</td>
<td>PMPIDR2</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
The PMPIDR3 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is **IMPLEMENTATION DEFINED** whether PMPIDR3 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is **OPTIONAL**.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMPIDR3 is a 32-bit register.

**Field descriptions**

The PMPIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>REVAND</td>
</tr>
<tr>
<td>29</td>
<td>CMOD</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**REVAND, bits [7:4]**

Part minor revision. Parts using PMPIDR2.REVISION as an extension to the Part number must use this field as a major revision number.

**CMOD, bits [3:0]**

Customer modified. Indicates someone other than the Designer has modified the component.

**Accessing the PMPIDR3**

PMPIDR3 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFEC</td>
<td>PMPIDR3</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
The PMPIDR4 characteristics are:

**Purpose**

Provides information to identify a Performance Monitor component.

For more information see 'About the Peripheral identification scheme' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section H8 (About the External Debug Registers).

**Configuration**

It is implementation defined whether PMPIDR4 is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is optional.

If ARMv8.3-DoPD is implemented, this register is in the Core power domain. If ARMv8.3-DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

**Attributes**

PMPIDR4 is a 32-bit register.

**Field descriptions**

The PMPIDR4 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | SIZE | DES_2 |

**Bits [31:8]**

Reserved, RES0.

**SIZE, bits [7:4]**

Size of the component. RAZ. Log2 of the number of 4KB pages from the start of the component to the end of the component ID registers.

**DES_2, bits [3:0]**

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is \(0b0100\).

**Accessing the PMPIDR4**

PMPIDR4 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xFD0</td>
<td>PMPIDR4</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When ARMv8.3-DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.
The PMSWINC_EL0 characteristics are:

**Purpose**

Increments a counter that is configured to count the Software increment event, event 0x00. For more information, see ‘SW_INCR’ in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section D5.

**Configuration**

External register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch64 System register PMSWINC_EL0[31:0].

External register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register PMSWINC[31:0].

PMSWINC_EL0 is in the Core power domain.

Implementation of this register is **OPTIONAL**.

If this register is implemented, use of it is deprecated.

If 1 is written to bit [n] from the external debug interface, it is **CONSTRAINED UNPREDICTABLE** whether or not a SW_INCR event is created for counter n. This is consistent with not implementing the register in the external debug interface.

**Attributes**

PMSWINC_EL0 is a 32-bit register.

**Field descriptions**

The PMSWINC_EL0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td><strong>RES0</strong></td>
</tr>
<tr>
<td>30</td>
<td><strong>P&lt;n&gt;, bit [n]</strong></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

-P<n>, bit [n], for n = 0 to 30

Event counter software increment bit for PMEVCNTR<n>_EL0.

If PMCFGR.N is less than 31, bits [30:PMCFGR.N] are WI.

<table>
<thead>
<tr>
<th>P&lt;n&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No action. The write to this bit is ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>It is <strong>CONSTRAINED UNPREDICTABLE</strong> whether a SW_INCR event is generated for event counter n.</td>
</tr>
</tbody>
</table>

**Accessing the PMSWINC_EL0**

**Note**
SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

**PMSWINC_EL0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0xCA0</td>
<td>PMSWINC_EL0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **WI**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.
PMVIDSR, VMID Sample Register

The PMVIDSR characteristics are:

**Purpose**

Contains the sampled VMID value that is captured on reading PMPCSR[31:0].

**Configuration**

PMVIDSR is in the Core power domain. RW fields in this register reset to architecturally UNKNOWN values. These apply only on a Cold reset. The register is not affected by a Warm reset and is not affected by an External debug reset.

This register is present only when ARMv8.2-PCSample is implemented and HaveEL(EL2). Otherwise, direct accesses to PMVIDSR are RES0.

---

**Note**

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of EDDEVID.PCSample.

---

**Attributes**

PMVIDSR is a 32-bit register.

**Field descriptions**

The PMVIDSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>VMID[15:8]</td>
</tr>
<tr>
<td>29</td>
<td>VMID</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>VMID</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>VMID</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>VMID</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>VMID</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>VMID</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>VMID</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>VMID</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>VMID</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>VMID</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>VMID</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>VMID</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>VMID</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>VMID</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>VMID</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**VMID[15:8], bits [15:8]**

When ARMv8.1-VMID16 is implemented:

Extension to VMID[7:0]. See VMID[7:0] for more details.

On a Cold reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**VMID, bits [7:0]**

VMID sample. The VMID associated with the most recent PMPCSR sample. When the most recent PMPCSR sample was generated:

- This field is set to an UNKNOWN value if any of the following apply:
  - EL2 is disabled in the current Security state.
The PE is executing at EL2.
- EL2 is enabled in the current Security state, the PE is executing at EL0, EL2 is using AArch64, HCR_EL2.E2H == 1, and HCR_EL2.TGE == 1.

Otherwise:
- If EL2 is using AArch64 and either ARMv8.1-VMID16 is not implemented or VTCR_EL2.VS is 1, this field is set to VTTBR_EL2.VMID.
- If EL2 is using AArch64, ARMv8.1-VMID16 is implemented, and VTCR_EL2.VS is 0, PMVIDSR.VMID[7:0] is set to VTTBR_EL2.VMID[7:0] and PMVIDSR.VMID[15:8] is RES0.
- If EL2 is using AArch32, this field is set to VTTBR_VMID.

Because the value written to PMVIDR is an indirect read of the VMID value, it is constrained unpredictable whether PMVIDSR is set to the original or new value if PMPCSR samples:

- An instruction that writes to the VMID value.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

On a Cold reset, this field resets to an architecturally unknown value.

## Accessing the PMVDSR

Implementation defined extensions to external debug might make the value of this register unknown, see 'Permitted behavior that might make the PC Sample-based profiling registers unknown' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile.

**PMVDSR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
<th>Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>PMU</td>
<td>0x20C</td>
<td>PMVDSR</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are RO.
- Otherwise accesses to this register generate an error response.
TRCACATR<n>, Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

**Purpose**

Defines the type of access for the corresponding TRCACVR<n> Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

**Configuration**

External register TRCACATR<n> bits [63:0] are architecturally mapped to AArch64 System register TRCACATR<n>[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACATR<n> are RES0.

**Attributes**

TRCACATR<n> is a 64-bit register.

**Field descriptions**

The TRCACATR<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | RES0 | EXLEVEL_NS_EL2 | EXLEVEL_NS_EL1 | EXLEVEL_NS_EL0 | EXLEVEL_S_EL3 | EXLEVEL_S_EL2 | EXLEVEL_S_EL1 | EXLEVEL_S_EL0 | RES0 | CONTEXT | CONTEXTTYPE | RES0 |
| 313029282726524232221201918171615 | 14 | 13 | 12 | 11 | 10 |

**Bits [63:15]**

Reserved, RES0.

**EXLEVEL_NS_EL2, bit [14]**

When Non-secure EL2 is implemented:

Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Non-secure EL2.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Non-secure EL2.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
# EXLEVEL_NS_EL1, bit [13]

When Non-secure EL1 is implemented:

Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Non-secure EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Non-secure EL1.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

# EXLEVEL_NS_EL0, bit [12]

When Non-secure EL0 is implemented:

Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Non-secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Non-secure EL0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

# EXLEVEL_S_EL3, bit [11]

When HaveEL(EL3):

Secure EL3 address comparison control. Controls whether a comparison can occur at EL3 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL3.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL3.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

# EXLEVEL_S_EL2, bit [10]

When HaveEL(EL2) and ARMv8.4-SecEL2 is implemented:

Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.
The Address Comparator performs comparisons in Secure EL2.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**EXLEVEL_S_EL1, bit [9]**

When Secure EL1 is implemented:

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL1.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL1.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**EXLEVEL_S_EL0, bit [8]**

When Secure EL0 is implemented:

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Comparator performs comparisons in Secure EL0.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Comparator does not perform comparisons in Secure EL0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bit [7]**

Reserved, RES0.

**CONTEXT, bits [6:4]**

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:
The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

If TRCIDR4(NUMCIDC == 0b0000 and TRCIDR4(NUMVMIDC == 0b0000, then this field is RES0.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**CONTEXTTYPE, bits [3:2]**

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons:

<table>
<thead>
<tr>
<th>CONTEXTTYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b01</td>
<td>The Address Comparator is dependent on the Context Identifier Comparator that the CONTEXT field specifies. If both the Context Identifier Comparator and the address comparison match, the Address Comparator signals a match.</td>
</tr>
<tr>
<td>0b10</td>
<td>The Address Comparator is dependent on the Virtual Context Identifier Comparator that the CONTEXT field specifies. If both the Virtual Context Identifier Comparator and the address comparison match, the Address Comparator signals a match.</td>
</tr>
<tr>
<td>0b11</td>
<td>The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that the CONTEXT field specifies. If the Context Identifier Comparator, the Virtual Context Identifier Comparator and address comparison all match, the Address Comparator signals a match.</td>
</tr>
</tbody>
</table>

If TRCIDR4(NUMCIDC == 0b0000, then bit [2] is RES0.

If TRCIDR4(NUMVMIDC == 0b0000, then bit [3] is RES0.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [1:0]**

Reserved, RES0.

**Accessing the TRCACATR<n>**

Must be programmed if any of the following are true:

- TRCBBCTRL.RANGE[n/2] == 0b1.
- TRCRSCTRL<a>.GROUP == 0b0100 and TRCRSCTRL<a>.SAC[n] == 0b1.
- TRCRSCTRL<a>.GROUP == 0b0101 and TRCRSCTRL<a>.ARC[n/2] == 0b1.
- TRCVIECTRL.EXCLUDE[n/2] == 0b1.
- TRCVIECTRL.INCLUDE[n/2] == 0b1.
• TRCVISSCTL, .START[n] == 0b1.
• TRCVISSCTL, .STOP[n] == 0b1.
• TRCSSCCR<>.ARC[n/2] == 0b1.
• TRCSSCCR<>.SAC[n] == 0b1.
• TRCOCTLR, .RANGE[n/2] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCACATR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x480 + 8n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
• Otherwise accesses to this register are **RW**.
The TRCACVR<n> characteristics are:

**Purpose**

Contains the address value.

**Configuration**

External register TRCACVR<n> bits [63:0] are architecturally mapped to AArch64 System register TRCACVR<n>[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACVR<n> are RES0.

**Attributes**

TRCACVR<n> is a 64-bit register.

**Field descriptions**

The TRCACVR<n> bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| ADDRESS |
| ADDRESS |

**ADDRESS, bits [63:0]**

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in the TRCACVR<n>. This requires that the trace analyzer always programs all implemented bits of the TRCACVR<n>.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as the virtual address size supported by the PE.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCACVR<n>**

Must be programmed if any of the following are true:

- TRCBBCTL.RANGE[n/2] == 0b1.
- TRCRSCTL<a>.GROUP == 0b0100 and TRCRSCTL<a>.SAC[n] == 0b1.
• TRCRSCTRL<a>.GROUP == 0b101 and TRCRSCTRL<a>.ARC[n/2] == 0b1.
• TRCVIIECTLR.EXCLUDE[n/2] == 0b1.
• TRCVIIECTLR.INCLUDE[n/2] == 0b1.
• TRCVISSCTLR.START[n] == 0b1.
• TRCVISSCTLR.STOP[n] == 0b1.
• TRCQCSSSCTR<>.ARC[n/2] == 0b1.
• TRCQCSSSCTR<>.SAC[n] == 0b1.
• TRCSSCTRL.RANGE[n/2] == 0b1.

Writes are constrained unpredictable if the trace unit is not in the Idle state.

**TRCACVR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x400 + 8n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
• Otherwise accesses to this register are RW.
TRCAUTHSTATUS, Authentication Status Register

The TRCAUTHSTATUS characteristics are:

**Purpose**

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information see the CoreSight Architecture Specification.

**Configuration**

External register TRCAUTHSTATUS bits [31:0] are architecturally mapped to AArch64 System register TRCAUTHSTATUS[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are RES0.

**Attributes**

TRCAUTHSTATUS is a 32-bit register.

**Field descriptions**

The TRCAUTHSTATUS bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>SNID</td>
</tr>
<tr>
<td>29</td>
<td>SID</td>
</tr>
<tr>
<td>28</td>
<td>NSNID</td>
</tr>
<tr>
<td>27</td>
<td>NSID</td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SNID, bits [7:6]**

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

<table>
<thead>
<tr>
<th>SNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Secure non-invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When EL3 is implemented, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b00.

When EL3 is not implemented and the PE is Secure, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

**SID, bits [5:4]**

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.
### SID, Meanings

<table>
<thead>
<tr>
<th>SID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Secure invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

### NSNID, Bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

<table>
<thead>
<tr>
<th>NSNID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-secure non-invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When EL3 is implemented, this field reads as 0b11.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b11.

When EL3 is not implemented and the PE is Secure, this field reads as 0b00.

### NSID, Bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

<table>
<thead>
<tr>
<th>NSID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Non-secure invasive debug features not implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Implemented and disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Implemented and enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

### Accessing the TRCAUTHSTATUS

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

External debugger accesses to this register are unaffected by the OS Lock.

**TRCAUTHSTATUS can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFB8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
The TRCAUXCTLR characteristics are:

**Purpose**

The function of this register is **IMPLEMENTATION DEFINED**.

**Configuration**

External register TRCAUXCTLR bits [31:0] are architecturally mapped to AArch64 System register `TRCAUXCTLR[31:0]`.

Some or all RW fields of this register have defined reset values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCAUXCTLR are RES0.

**Attributes**

TRCAUXCTLR is a 32-bit register.

**Field descriptions**

The TRCAUXCTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit Position</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-0</td>
<td>IMPLEMENTATION DEFINED</td>
</tr>
</tbody>
</table>

**IMPLEMENTATION DEFINED, bits [31:0]**

**IMPLEMENTATION DEFINED.**

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

On a Trace unit reset, this field resets to 0.

**Accessing the TRCAUXCTLR**

If this register is set to nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

TRCAUXCTLR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x018</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
# TRCBBCTLR, Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

## Purpose

Controls the regions in the memory map where branch broadcasting is active.

## Configuration

External register TRCBBCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCBBCTLR[31:0]. RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ETE is implemented, TRCIDR0.TRCBB == 0b1 and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCBBCTLR are **RES0**.

## Attributes

TRCBBCTLR is a 32-bit register.

## Field descriptions

The TRCBBCTLR bit assignments are:

<table>
<thead>
<tr>
<th>RES0</th>
<th>MODE</th>
<th>RANGE&lt;0..7&gt;</th>
<th>bit [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:9]**

Reserved, **RES0**.

**MODE, bit [8]**

Mode.

<table>
<thead>
<tr>
<th>MODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by RANGE. If RANGE == 0x00 then branch broadcasting is active for all instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>Include Mode. Branch broadcasting is active for instructions in the address ranges defined by RANGE. If RANGE == 0x00 then the behavior of the trace unit is <strong>CONSTRAINED UNPREDICTABLE</strong>. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**RANGE<0..7>, bit [m], for m = 0 to 7**

Address range field.

Selects which Address Range Comparators are in use with branch broadcasting.
<table>
<thead>
<tr>
<th>RANGE&lt;\text{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address range that Address Range Comparator \text{m} defines, is not selected.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address range that Address Range Comparator \text{m} defines, is selected.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \text{m} \geq \text{TRCIDR4.NUMACPAIRS}.

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

### Accessing the TRCBBCTL

Must be programmed if \text{TRCCONFIGR.BB} == 0b1.

Writes are \texttt{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

**TRCBBCTL can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x03C</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \text{!AllowExternalTraceAccess()}, or \text{OSLockStatus()} or \text{!IsTraceCorePowered()} accesses to this register generate an error response.
- Otherwise accesses to this register are \texttt{RW}.
TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR characteristics are:

**Purpose**

Set the threshold value for cycle counting.

**Configuration**

External register TRCCCCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCCCCTLR[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ETE is implemented and TRCIDR0.TRCCCI == 0b1. Otherwise, direct accesses to TRCCCCTLR are **RES0**.

**Attributes**

TRCCCCTLR is a 32-bit register.

**Field descriptions**

The TRCCCCTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>THRESHOLD</td>
</tr>
</tbody>
</table>

**Bits [31:12]**

Reserved, RES0.

**THRESHOLD, bits [11:0]**

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in TRCIDR3.CCITMIN. If the THRESHOLD value is smaller than the value in TRCIDR3.CCITMIN then the behavior is **CONSTRAINED** **UNPREDICTABLE**. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when TRCONFIGR.CCI is set to enable instruction trace cycle counting, results in **CONSTRAINED** **UNPREDICTABLE** behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Accessing the TRCCCCTLR**

Must be programmed if TRCONFIGR.CCI == 0b1.

Writes are **CONSTRAINED** **UNPREDICTABLE** if the trace unit is not in the Idle state.

**TRCCCCTLR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x038</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
The TRCCIDCCTRL0 characteristics are:

**Purpose**

Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 0 to 3.

**Configuration**

External register TRCCIDCCTRL0 bits [31:0] are architecturally mapped to AArch64 System register TRCCIDCCTRL0[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented, TRCIDR4.NUMCIDC > 0x0 and TRCIDR2.CIDSIZE > 0b00000. Otherwise, direct accesses to TRCCIDCCTRL0 are RES0.

**Attributes**

TRCCIDCCTRL0 is a 32-bit register.

**Field descriptions**

The TRCCIDCCTRL0 bit assignments are:

<table>
<thead>
<tr>
<th>Field</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP3&lt;m&gt;, bit</td>
<td>The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the</td>
</tr>
<tr>
<td>[m+24]</td>
<td>Context identifier comparison.</td>
</tr>
<tr>
<td>COMP2&lt;m&gt;, bit</td>
<td>The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the</td>
</tr>
<tr>
<td>[m+16]</td>
<td>Context identifier comparison.</td>
</tr>
<tr>
<td>COMP1&lt;m&gt;, bit</td>
<td>The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the</td>
</tr>
<tr>
<td>[m+8]</td>
<td>Context identifier comparison.</td>
</tr>
<tr>
<td>COMP0&lt;m&gt;, bit</td>
<td>The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the</td>
</tr>
<tr>
<td>[m]</td>
<td>Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**COMP2<m>, bit [m+16], for m = 0 to 7**

When TRCIDR4.NUMCIDC > 2:

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.
TRCCIDCCTL0, Context Identifier Comparator Control Register 0

<table>
<thead>
<tr>
<th>COMP2&lt;\text{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \( m \geq \text{TRCIDR2.CIDSIZE} \).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

**COMP1<\text{m}>, bit [\text{m}+8], for \text{m} = 0 to 7**

When \text{TRCIDR4.NUMCIDC} > 1:

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

<table>
<thead>
<tr>
<th>COMP1&lt;\text{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \( m \geq \text{TRCIDR2.CIDSIZE} \).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

**COMP0<\text{m}>, bit [\text{m}], for \text{m} = 0 to 7**

When \text{TRCIDR4.NUMCIDC} > 0:

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

<table>
<thead>
<tr>
<th>COMP0&lt;\text{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \( m \geq \text{TRCIDR2.CIDSIZE} \).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

**Accessing the TRCCIDCCTL0**

If software uses the TRCCIDCVR<n> registers, for \( n = 0 \) to 3, then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCCIDCVR<n> to 0x00.
If any bit is 0b1 and the relevant byte in TRCCIDCVR<n> is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCCIDCCTRLR0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x680</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
The TRCCIDCCTRL1 characteristics are:

**Purpose**
Contains Context identifier mask values for the TRCCIDCVR<n> registers, for n = 4 to 7.

**Configuration**
External register TRCCIDCCTRL1 bits [31:0] are architecturally mapped to AArch64 System register TRCCIDCCTRL1[31:0].
RW fields in this register reset to architecturally UNKNOWN values.
This register is present only when ETE is implemented, TRCIDR4.NUMCIDC > 0x4 and TRCIDR2.CIDSIZE > 0b00000. Otherwise, direct accesses to TRCCIDCCTRL1 are RES0.

**Attributes**
TRCCIDCCTRL1 is a 32-bit register.

**Field descriptions**
The TRCCIDCCTRL1 bit assignments are:

<table>
<thead>
<tr>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
<th>COMP7&lt;m&gt;, bit [m+24]</th>
<th>COMP6&lt;m&gt;, bit [m+16]</th>
<th>COMP5&lt;m&gt;, bit [m+8]</th>
<th>COMP4&lt;m&gt;, bit [m]</th>
</tr>
</thead>
</table>

**COMP7<m>, bit [m+24], for m = 0 to 7**
When TRCIDR4.NUMCIDC > 7:
TRCCIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR7. Each bit in this field corresponds to a byte in TRCCIDCVR7.

<table>
<thead>
<tr>
<th>COMP7&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.
On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**
Reserved, RES0.

**COMP6<m>, bit [m+16], for m = 0 to 7**
When TRCIDR4.NUMCIDC > 6:
TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.
<table>
<thead>
<tr>
<th>COMP6&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**COMP5<m>, bit [m+8], for m = 0 to 7**

**When TRCIDR4.NUMCIDC > 5:**

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

<table>
<thead>
<tr>
<th>COMP5&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**COMP4<m>, bit [m], for m = 0 to 7**

**When TRCIDR4.NUMCIDC > 4:**

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

<table>
<thead>
<tr>
<th>COMP4&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR2.CIDSIZE.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Accessing the TRCCIDCCTRL1**

If software uses the TRCCIDCVR<n> registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCCIDCVR<n> to 0x00.
If any bit is 0b1 and the relevant byte in TRCCIDCVR<n> is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCCIDCCTRL1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x684</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
The TRCCIDCVR\(n\) characteristics are:

**Purpose**

Contains a Context identifier value.

**Configuration**

External register TRCCIDCVR\(n\) bits [63:0] are architecturally mapped to AArch64 System register TRCCIDCVR\(n\)[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMCIDC > n. Otherwise, direct accesses to TRCCIDCVR\(n\) are RES0.

**Attributes**

TRCCIDCVR\(n\) is a 64-bit register.

**Field descriptions**

The TRCCIDCVR\(n\) bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>63</td>
<td>62</td>
</tr>
<tr>
<td>61</td>
<td>60</td>
</tr>
<tr>
<td>59</td>
<td>58</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
</tr>
<tr>
<td>55</td>
<td>54</td>
</tr>
<tr>
<td>53</td>
<td>52</td>
</tr>
<tr>
<td>51</td>
<td>50</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
</tr>
<tr>
<td>47</td>
<td>46</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
</tr>
<tr>
<td>43</td>
<td>42</td>
</tr>
<tr>
<td>41</td>
<td>40</td>
</tr>
<tr>
<td>39</td>
<td>38</td>
</tr>
<tr>
<td>37</td>
<td>36</td>
</tr>
<tr>
<td>35</td>
<td>34</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>31</td>
<td>30</td>
</tr>
<tr>
<td>29</td>
<td>28</td>
</tr>
<tr>
<td>27</td>
<td>26</td>
</tr>
<tr>
<td>25</td>
<td>24</td>
</tr>
<tr>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
</tr>
<tr>
<td>19</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**VALUE**, bits [63:0]

Context identifier value. The width of this field is indicated by TRCIDR2.CIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCCIDCVR\(n\)**

Must be programmed if any of the following are true:

- TRCRSCTLR\(a\).GROUP == \(0b110\) and TRCRSCTLR\(a\).CID\(n\) == \(0b1\).
- TRCACATR\(a\).CONTEXTTYPE == \(0b01\) or \(0b11\) and TRCACATR\(a\).CONTEXT == \(n\).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCCIDCVR\(n\) can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x600 + (8n)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
- When `!AllowExternalTraceAccess()`, or `OSLockStatus()` or `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
The TRCCIDR0 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCIDR0 are RES0.

There are no configuration notes.

**Attributes**

TRCCIDR0 is a 32-bit register.

**Field descriptions**

The TRCCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit positions</th>
<th>Bit assignments</th>
</tr>
</thead>
<tbody>
<tr>
<td>31:8</td>
<td>RES0</td>
</tr>
<tr>
<td>7:0</td>
<td>PRMBL_0</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_0, bits [7:0]**

Component identification preamble, segment 0.

This field reads as 0x0D.

**Accessing the TRCCIDR0**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCCIDR0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFF0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCCIDR1, Component Identification Register 1

The TRCCIDR1 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCIDR1 are RES0.

There are no configuration notes.

**Attributes**

TRCCIDR1 is a 32-bit register.

**Field descriptions**

The TRCCIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>CLASS</td>
</tr>
<tr>
<td>29</td>
<td>PRMBL_1</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**CLASS, bits [7:4]**

Component class.

<table>
<thead>
<tr>
<th>CLASS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1001</td>
<td>CoreSight peripheral.</td>
</tr>
</tbody>
</table>

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

**PRMBL_1, bits [3:0]**

Component identification preamble, segment 1.

This field reads as 0x0.

**Accessing the TRCCIDR1**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCCIDR1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.
The TRCCIDR2 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCIDR2 are RES0.

There are no configuration notes.

**Attributes**

TRCCIDR2 is a 32-bit register.

**Field descriptions**

The TRCCIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>8</td>
<td>PRMBL_2, bits [7:0]</td>
</tr>
</tbody>
</table>

- **Bits [31:8]**
  - Reserved, RES0.
- **PRMBL_2, bits [7:0]**
  - Component identification preamble, segment 2.
  - This field reads as 0x05.

**Accessing the TRCCIDR2**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCCIDR2 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFF8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCCIDR3, Component Identification Register 3

The TRCCIDR3 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCIDR3 are RES0.

There are no configuration notes.

**Attributes**

TRCCIDR3 is a 32-bit register.

**Field descriptions**

The TRCCIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>PRMBL_3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PRMBL_3, bits [7:0]**

Component identification preamble, segment 3.

This field reads as 0xB1.

**Accessing the TRCCIDR3**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCCIDR3 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFFC</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
The TRCCLAIMCLR characteristics are:

**Purpose**

In conjunction with TRCCLAIMSET, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information see the CoreSight Architecture Specification.

**Configuration**

External register TRCCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register TRCCLAIMCLR[31:0].

Some or all RW fields of this register have defined reset values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCLAIMCLR are RES0.

**Attributes**

TRCCLAIMCLR is a 32-bit register.

**Field descriptions**

The TRCCLAIMCLR bit assignments are:

<table>
<thead>
<tr>
<th>CLR&lt;m&gt;, bit [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**CLR<m>, bit [m], for m = 0 to 31**

Claim Tag Clear. Indicates the current status of the Claim Tag bit m, and is used to clear Claim Tag bit m to 0b0.

<table>
<thead>
<tr>
<th>CLR&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>On a read: Claim Tag bit m is not set. On a write: Ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>On a read: Claim Tag bit m is set. On a write: Clear Claim tag bit m to 0b0.</td>
</tr>
</tbody>
</table>

The number of Claim Tag bits implemented is indicated in TRCCLAIMSET.

On a Trace unit reset, this field resets to 0.

### Accessing the TRCCLAIMCLR

TRCCLAIMCLR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFA4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCCLAIMSET, Claim Tag Set Register

The TRCCLAIMSET characteristics are:

**Purpose**

In conjunction with TRCCLAIMCLR, provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information see the CoreSight Architecture Specification.

**Configuration**

External register TRCCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register TRCCLAIMSET[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCLAIMSET are RES0.

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as 0b1111.

**Attributes**

TRCCLAIMSET is a 32-bit register.

**Field descriptions**

The TRCCLAIMSET bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| **SET<m>**, bit [m] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

SET<m>, bit [m], for m = 0 to 31

Claim Tag Set. Indicates whether Claim Tag bit m is implemented, and is used to set Claim Tag bit m to 0b1.

<table>
<thead>
<tr>
<th>SET&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>On a read: Claim Tag bit m is not implemented. On a write: Ignored.</td>
</tr>
<tr>
<td>0b1</td>
<td>On a read: Claim Tag bit m is implemented. On a write: Set Claim Tag bit m to 0b1.</td>
</tr>
</tbody>
</table>

**Accessing the TRCCLAIMSET**

TRCCLAIMSET can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFA0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCCNTCTLR<n>, Counter Control Register <n>, n = 0 - 3

The TRCCNTCTLR<n> characteristics are:

**Purpose**

Controls the operation of Counter <n>.

**Configuration**

External register TRCCNTCTLR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCCNTCTLR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTCTLR<n> are RES0.

**Attributes**

TRCCNTCTLR<n> is a 32-bit register.

**Field descriptions**

The TRCCNTCTLR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | CNTCHAIN | RLDSELF | RLDEVENT_TYPE | RES0 | RLDEVENT_SEL | CNTEVENT_TYPE | RES0 | CNTEVENT_SEL | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 | RES0 |

**Bits [31:18]**

Reserved, RES0.

**CNTCHAIN, bit [17]**

For TRCCNTCTLR3 and TRCCNTCTLR1, this bit controls whether the Counter decrements when a reload event occurs for Counter <n-1>.

<table>
<thead>
<tr>
<th>CNTCHAIN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Counter does not decrement when a reload event for Counter &lt;n-1&gt; occurs.</td>
</tr>
<tr>
<td>0b1</td>
<td>Counter &lt;n&gt; decrements when a reload event for Counter &lt;n-1&gt; occurs. This concatenates Counter &lt;n&gt; and Counter &lt;n-1&gt;, to provide a larger count value.</td>
</tr>
</tbody>
</table>

CNTCHAIN is not implemented for TRCCNTCTLR0 and TRCCNTCTLR2.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**RLDSELF, bit [16]**

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.
### RLDSEL
<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 Normal mode.</td>
</tr>
<tr>
<td>The Counter is in Normal mode.</td>
</tr>
<tr>
<td>0b1 Self-reload mode.</td>
</tr>
<tr>
<td>The Counter is in Self-reload mode.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### RLDEVENT_TYPE, bit [15]

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes a reload event for Counter <n>.

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 A single Resource Selector:</td>
</tr>
<tr>
<td>TRCCNTCTLR&lt;n&gt;.RLDEVENT.SEL[4:0] selects</td>
</tr>
<tr>
<td>the single Resource Selector, from 0-31,</td>
</tr>
<tr>
<td>used to activate the resource event.</td>
</tr>
<tr>
<td>0b1 A Boolean-combined pair of Resource</td>
</tr>
<tr>
<td>Selectors. TRCCNTCTLR&lt;n&gt;.RLDEVENT.SEL[3:0]</td>
</tr>
<tr>
<td>selects the Resource Selector pair, from</td>
</tr>
<tr>
<td>0-15, that has a Boolean function that is</td>
</tr>
<tr>
<td>applied to it whose output is used to</td>
</tr>
<tr>
<td>activate the resource event.</td>
</tr>
<tr>
<td>TRCCNTCTLR&lt;n&gt;.RLDEVENT.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### Bits [14:13]

Reserved, RES0.

### RLDEVENT_SEL, bits [12:8]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.TYPE controls whether TRCCNTCTLR<n>.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes a reload event for Counter <n>.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is **UNPREDICTABLE**, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

### CNTEVENT_TYPE, bit [7]

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes Counter <n> to decrement.

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0 A single Resource Selector:</td>
</tr>
<tr>
<td>TRCCNTCTLR&lt;n&gt;.CNTEVENT.SEL[4:0] selects</td>
</tr>
<tr>
<td>the single Resource Selector, from 0-31,</td>
</tr>
<tr>
<td>used to activate the resource event.</td>
</tr>
<tr>
<td>0b1 A Boolean-combined pair of Resource</td>
</tr>
<tr>
<td>Selectors. TRCCNTCTLR&lt;n&gt;.CNTEVENT.SEL[3:0]</td>
</tr>
<tr>
<td>selects the Resource Selector pair, from</td>
</tr>
<tr>
<td>0-15, that has a Boolean function that is</td>
</tr>
<tr>
<td>applied to it whose output is used to</td>
</tr>
<tr>
<td>activate the resource event.</td>
</tr>
<tr>
<td>TRCCNTCTLR&lt;n&gt;.CNTEVENT.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.
Bits [6:5]

Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTRLR\textsubscript{n}.CNTEVENT.TYPE controls whether TRCCNTCTRLR\textsubscript{n}.CNTEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes Counter \textsubscript{n} to decrement.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCCNTCTRLR\textsubscript{n}

Must be programmed if TRCRSCTRLR\textsubscript{a}.GROUP == 0b0010 and TRCRSCTRLR\textsubscript{a}.COUNTERS\[n\] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCNTCTRLR\textsubscript{n} can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x150 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCCNTRLDVR<n>, Counter Reload Value Register <n>, n = 0 - 3

The TRCCNTRLDVR<n> characteristics are:

**Purpose**

This sets or returns the reload count value for Counter <n>.

**Configuration**

External register TRCCNTRLDVR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCCNTRLDVR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTRLDVR<n> are RES0.

**Attributes**

TRCCNTRLDVR<n> is a 32-bit register.

**Field descriptions**

The TRCCNTRLDVR<n> bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>VALUE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**VALUE, bits [15:0]**

Contains the reload value for Counter <n>. When a reload event occurs for Counter <n> then the trace unit copies the VALUE<n> field into Counter <n>.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCCNTRLDVR<n>**

Must be programmed if TRCRSCTRLR<a>.GROUP == 0b0010 and TRCRSCTRLR<a>.COUNTERS[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCCNTRLDVR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x140 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When `!AllowExternalTraceAccess()`, or `OSLockStatus()` or `!IsTraceCorePowered()` accesses to this register generate an error response.
• Otherwise accesses to this register are **RW**.
The TRCCNTVR<n> characteristics are:

**Purpose**

This sets or returns the value of Counter <n>.

**Configuration**

External register TRCCNTVR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCCNTVR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTVR<n> are RES0.

**Attributes**

TRCCNTVR<n> is a 32-bit register.

**Field descriptions**

The TRCCNTVR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-</td>
<td>RES0</td>
</tr>
<tr>
<td>15-</td>
<td>VALUE</td>
</tr>
</tbody>
</table>

**Bits [31:16]**

Reserved, RES0.

**VALUE, bits [15:0]**

Contains the count value of Counter.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCCNTVR<n>**

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.COUNTERS[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

**TRCCNTVR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x160 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

**Purpose**

Controls the tracing options.

**Configuration**

External register TRCCONFIGR bits [31:0] are architecturally mapped to AArch64 System register TRCCONFIGR[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCCONFIGR are **RES0**.

**Attributes**

TRCCONFIGR is a 32-bit register.

**Field descriptions**

The TRCCONFIGR bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

<table>
<thead>
<tr>
<th></th>
<th>RES0</th>
<th>RES0</th>
<th>QE</th>
<th>RS</th>
<th>TS</th>
<th>RES0</th>
<th>VMID</th>
<th>CID</th>
<th>RES0</th>
<th>BB</th>
<th>RES0</th>
<th>RES1</th>
</tr>
</thead>
</table>

**Bits [31:16]**

Reserved, **RES0**.

**Bit [15]**

When TRCIDR2.VMIDOPT == 0b00:

Reserved, **RES0**.

Virtual context identifier selection control.

**VTBTR_EL2** VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b10:

Reserved, **RES1**.

Virtual context identifier selection control.

**CONTEXTIDR_EL2** PROCID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b01:

Virtual context identifier selection control.
### VMIDOPT

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>VTTBR_EL2: VMID is used as the Virtual context identifier.</td>
</tr>
<tr>
<td>0b1</td>
<td>CONTEXTIDR_EL2: PROCID is used as the Virtual context identifier.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

### QE, bits [14:13]

**When TRCIDR0.QSUPP == 0b01:**

Q element generation control.

<table>
<thead>
<tr>
<th>QE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q elements are disabled.</td>
</tr>
<tr>
<td>0b01</td>
<td>Q elements with instruction counts are enabled.</td>
</tr>
<tr>
<td></td>
<td>Q elements without instruction counts are disabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**When TRCIDR0.QSUPP == 0b10:**

Q element generation control.

<table>
<thead>
<tr>
<th>QE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q elements are disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Q elements with instruction counts are enabled.</td>
</tr>
<tr>
<td></td>
<td>Q elements without instruction counts are enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**When TRCIDR0.QSUPP == 0b11:**

Q element generation control.

<table>
<thead>
<tr>
<th>QE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q elements are disabled.</td>
</tr>
<tr>
<td>0b01</td>
<td>Q elements with instruction counts are enabled.</td>
</tr>
<tr>
<td></td>
<td>Q elements without instruction counts are disabled.</td>
</tr>
<tr>
<td>0b11</td>
<td>Q elements with instruction counts are enabled.</td>
</tr>
<tr>
<td></td>
<td>Q elements without instruction counts are enabled.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

### RS, bit [12]

**When TRCIDR0.RETSTACK == 0b1:**

Return stack control.

<table>
<thead>
<tr>
<th>RS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Return stack is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Return stack is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

**TS, bit [11]**

When TRCIDR0.TSSIZE != 0b00000:

Global timestamp tracing control.

<table>
<thead>
<tr>
<th>TS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Global timestamp tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Global timestamp tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [10:8]**

Reserved, RES0.

**VMID, bit [7]**

When TRCIDR2.VMIDSIZE != 0b00000:

Virtual context identifier tracing control.

<table>
<thead>
<tr>
<th>VMID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual context identifier tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual context identifier tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**CID, bit [6]**

When TRCIDR2.CIDSIZE != 0b00000:

Context identifier tracing control.

<table>
<thead>
<tr>
<th>CID</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Context identifier tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Context identifier tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bit [5]**

Reserved, RES0.
CCI, bit [4]

When TRCIDR0.TRCCCI == 0b1:

Cycle counting instruction tracing control.

<table>
<thead>
<tr>
<th>CCI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting instruction tracing is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting instruction tracing is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BB, bit [3]

When TRCIDR0.TRCBB == 0b1:

Branch broadcasting control.

<table>
<thead>
<tr>
<th>BB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Branch broadcasting is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Branch broadcasting is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [2:1]

Reserved, RES0.

Bit [0]

Reserved, RES1.

Accessing the TRCCONFIGR

Must always be programmed.

TRCCONFIGR.QE must be set to 0b00 if TRCCONFIGR.BB is not 0b0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCONFIGR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x010</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCDEVAFF, Device Affinity Register

The TRCDEVAFF characteristics are:

**Purpose**

For additional information see the CoreSight Architecture Specification.

Reads the same value as the MPIDR_EL1 register for the PE that this trace unit has affinity with.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVAFF are RES0.

There are no configuration notes.

**Attributes**

TRCDEVAFF is a 64-bit register.

**Field descriptions**

The TRCDEVAFF bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

MPIDR_EL1, bits [63:0]

Read-only copy of MPIDR_EL1, as seen from the highest implemented Exception level.

**Accessing the TRCDEVAFF**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCDEVAFF can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFA8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCDEVARCH, Device Architecture Register

The TRCDEVARCH characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

External register TRCDEVARCH bits [31:0] are architecturally mapped to AArch64 System register TRCDEVARCH[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVARCH are RES0.

**Attributes**

TRCDEVARCH is a 32-bit register.

**Field descriptions**

The TRCDEVARCH bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
|     | ARCHITECT | PRESENT | REVISON | ARCHVER | ARCHPART |

**ARCHITECT, bits [31:21]**

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

<table>
<thead>
<tr>
<th>ARCHITECT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1001111011</td>
<td>JEP106 continuation code 0x4, ID code 0x3B. Arm Limited.</td>
</tr>
</tbody>
</table>

Other values are defined by the JEDEC JEP106 standard.

This field reads as 0x23B.

**PRESENT, bit [20]**

DEVARCH Present. Defines that the DEVARCH register is present.

<table>
<thead>
<tr>
<th>PRESENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Device Architecture information not present.</td>
</tr>
<tr>
<td>0b1</td>
<td>Device Architecture information present.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

**REVISION, bits [19:16]**

Revision. Defines the architecture revision of the component.

<table>
<thead>
<tr>
<th>REVISION</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>ETE Version 1.0.</td>
</tr>
</tbody>
</table>
All other values are reserved.

**ARCHVER, bits [15:12]**

Architecture Version. Defines the architecture version of the component.

<table>
<thead>
<tr>
<th>ARCHVER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0101</td>
<td>ETE Version 1.</td>
</tr>
</tbody>
</table>

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12]. This field reads as 0x5.

**ARCHPART, bits [11:0]**

Architecture Part. Defines the architecture of the component.

<table>
<thead>
<tr>
<th>ARCHPART</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA13</td>
<td>Arm PE trace architecture.</td>
</tr>
</tbody>
</table>

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0]. This field reads as 0xA13.

**Accessing the TRCDEVARCH**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCDEVARCH can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFBC</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCDEVID, Device Configuration Register

The TRCDEVID characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

External register TRCDEVID bits [31:0] are architecturally mapped to AArch64 System register TRCDEVID[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVID are RES0.

**Attributes**

TRCDEVID is a 32-bit register.

**Field descriptions**

The TRCDEVID bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |

**Bits [31:0]**

Reserved, RES0.

**Accessing the TRCDEVID**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCDEVID can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFC8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCDEVID1, Device Configuration Register 1

The TRCDEVID1 characteristics are:

**Purpose**

Provides discovery information for the component.
For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVID1 are \texttt{RES0}.
There are no configuration notes.

**Attributes**

TRCDEVID1 is a 32-bit register.

**Field descriptions**

The TRCDEVID1 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, \texttt{RES0}</td>
</tr>
</tbody>
</table>

**Accessing the TRCDEVID1**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCDEVID1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFC4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{!IsTraceCorePowered()} accesses to this register generate an error response.
- Otherwise accesses to this register are \texttt{RO}.
TRCDEVID2, Device Configuration Register 2

The TRCDEVID2 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVID2 are *RES0*.

There are no configuration notes.

**Attributes**

TRCDEVID2 is a 32-bit register.

**Field descriptions**

The TRCDEVID2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |

**Bits [31:0]**

Reserved, RES0.

**Accessing the TRCDEVID2**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCDEVID2 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFC0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCDEVTYPE, Device Type Register

The TRCDEVTYPE characteristics are:

**Purpose**

Provides discovery information for the component. If the part number field is not recognised, a debugger can report the information that is provided by TRCDEVTYPE about the component instead.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCDEVTYPE are RES0.

There are no configuration notes.

**Attributes**

TRCDEVTYPE is a 32-bit register.

**Field descriptions**

The TRCDEVTYPE bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>SUB</td>
<td>MAJOR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**SUB, bits [7:4]**

Component sub-type.

<table>
<thead>
<tr>
<th>SUB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0001</td>
<td>When MAJOR == 0x3 (Trace source): Associated with a PE.</td>
</tr>
</tbody>
</table>

This field reads as 0x1.

**MAJOR, bits [3:0]**

Component major type.

<table>
<thead>
<tr>
<th>MAJOR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0011</td>
<td>Trace source.</td>
</tr>
</tbody>
</table>

Other values are defined by the CoreSight Architecture.

This field reads as 0x3.

**Accessing the TRCDEVTYPE**

External debugger accesses to this register are unaffected by the OS Lock.
TRCDEVTYPE can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFCC</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCEVENTCTL0R, Event Control 0 Register

The TRCEVENTCTL0R characteristics are:

**Purpose**

Controls the generation of ETEEvents.

**Configuration**

External register TRCEVENTCTL0R bits [31:0] are architecturally mapped to AArch64 System register TRCEVENTCTL0R[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMRSPAIR != 0b0000. Otherwise, direct accesses to TRCEVENTCTL0R are RES0.

**Attributes**

TRCEVENTCTL0R is a 32-bit register.

**Field descriptions**

The TRCEVENTCTL0R bit assignments are:

<table>
<thead>
<tr>
<th>EVENT3_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT3.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [30:29]**

Reserved, RES0.

**EVENT3_SEL**, bits [28:24]

Reserved, RES0.
When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[3] == 0b1, then Event element 3 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturallyUNKNOWN value.

Otherwise:

Reserved, RES0.

**EVENT2_TYPE, bit [23]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT2_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturallyUNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [22:21]**

Reserved, RES0.

**EVENT2_SEL, bits [20:16]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[2] == 0b1, then Event element 2 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturallyUNKNOWN value.
Otherwise:

Reserved, RES0.

**EVENT1_TYPE, bit [15]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT1_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT1.SEL[4:0] selects the single Resource Selector; from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT1.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**Bits [14:13]**

Reserved, RES0.

**EVENT1_SEL, bits [12:8]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[1] == 0b1, then Event element 1 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

**EVENT0_TYPE, bit [7]**

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Chooses the type of Resource Selector.
<table>
<thead>
<tr>
<th>EVENT0_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCEVENTCTL0R.EVENT0_SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0_SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT0_SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES0.

### Bits [6:5]

Reserved, RES0.

### EVENT0_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0_SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

When any of the selected resource events occurs and TRCEVENTCTL1R.INSTEN[0] == 0b1, then Event element 0 is generated in the instruction trace element stream.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

### Otherwise:

Reserved, RES0.

### Accessing the TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

### TRCEVENTCTL0R can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x020</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCEVENTCTL1R, Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

**Purpose**

Controls the behavior of the ETEEvents that TRCEVENTCTL0R selects.

**Configuration**

External register TRCEVENTCTL1R bits [31:0] are architecturally mapped to AArch64 System register TRCEVENTCTL1R[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are RES0.

**Attributes**

TRCEVENTCTL1R is a 32-bit register.

**Field descriptions**

The TRCEVENTCTL1R bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>LPOVERRIDE</td>
</tr>
<tr>
<td>29</td>
<td>ATB</td>
</tr>
<tr>
<td>28</td>
<td>RES0</td>
</tr>
<tr>
<td>27</td>
<td>INSTEN&lt;m&gt;, bit [m]</td>
</tr>
</tbody>
</table>

**Bits [31:13]**

Reserved, RES0.

**LPOVERRIDE, bit [12]**

When TRCIDR5.LPOVERRIDE == 0b1:

Low-power Override Mode select.

<table>
<thead>
<tr>
<th>LPOVERRIDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**ATB, bit [11]**

When TRCIDR5.ATBTRIG == 0b1:

AMBA Trace Bus (ATB) trigger enable.

If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:
• ATID == 0x7D.
• ATDATA to the value of TRCTRACEIDR.

If the width of ATDATA is greater than the width of TRCTRACEIDR, TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

<table>
<thead>
<tr>
<th>ATB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>ATB trigger is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>ATB trigger is enabled.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**Bits [10:4]**

Reserved, RES0.

**INSTEN<m>, bit [m], for m = 0 to 3**

Event element control.

<table>
<thead>
<tr>
<th>INSTEN&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit does not generate an Event element m.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit generates an Event element m.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= the number indicated by TRCIDR0.NUMEVENT.

**Accessing the TRCEVENTCTL1R**

Must be programmed.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

**TRCEVENTCTL1R can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x024</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
TRCEXTINSELR<n>, External Input Select Register <n>, n = 0 - 3

The TRCEXTINSELR<n> characteristics are:

**Purpose**

Use this to set, or read, which External Inputs are resources to the trace unit.

**Configuration**

External register TRCEXTINSELR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCEXTINSELR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.NUMEXTINSEL > n. Otherwise, direct accesses to TRCEXTINSELR<n> are RES0.

**Attributes**

TRCEXTINSELR<n> is a 32-bit register.

**Field descriptions**

The TRCEXTINSELR<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | evtCount |

**Bits [31:16]**

Reserved, RES0.

**evtCount, bits [15:0]**

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range 0x0000 to 0x003F are common architectural and microarchitectural events.
- PMU event numbers in the range 0x0040 to 0x00BF are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range 0x00C0 to 0x3FF are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range 0x0000 to 0x003F, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.
Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCEXTINSELR<n>**

Must be programmed if any of the following is true: TRCRSCTLR<a>.GROUP == 0b0000 and TRCRSCTLR<a>.EXTIN[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCEXTINSELR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x120 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCIDR0, ID Register 0

The TRCIDR0 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR0 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR0[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR0 are reserved.

**Attributes**

TRCIDR0 is a 32-bit register.

**Field descriptions**

The TRCIDR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit (31)</th>
<th>Bit (30)</th>
<th>Bit (29)</th>
<th>Bit (28:24)</th>
<th>Bit (17:10)</th>
<th>Bit (9:8)</th>
<th>Bit (7:4)</th>
<th>Bit (3:0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>COMMTRANS</td>
<td>COMMOPT</td>
<td>TSIZE</td>
<td>RES0</td>
<td>TRCIDR8.MAXSPEC</td>
<td>TRCIDR8.MAXSPEC</td>
<td>TRCIDR8.MAXSPEC</td>
</tr>
</tbody>
</table>

**Bit [31]**

Reserved, RES0.

**COMMTRANS, bit [30]**

Transaction Start element behavior.

<table>
<thead>
<tr>
<th>COMMTRANS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Transaction Start elements are P0 elements.</td>
</tr>
<tr>
<td>0b1</td>
<td>Transaction Start elements are not P0 elements.</td>
</tr>
</tbody>
</table>

**COMMOPT, bit [29]**

Indicates how the commit field in Cycle count packets is interpreted.

<table>
<thead>
<tr>
<th>COMMOPT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Commit mode 0. Cycle count packets includes the Commit element to which the Cycle Count element is attached.</td>
</tr>
<tr>
<td>0b1</td>
<td>Commit mode 1. Cycle count packets do not include Commit elements. The Cycle Count element is attached to the most recent Commit element.</td>
</tr>
</tbody>
</table>

This bit reads-as-one if TRCIDR0.TRCCCI == 0b1 and TRCIDR8.MAXSPEC == 0x0. This bit reads-as-zero if TRCIDR0.TRCCCI == 0b0.

**TSIZE, bits [28:24]**

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.
<table>
<thead>
<tr>
<th>TSSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Global timestamping not implemented.</td>
</tr>
<tr>
<td>0b01000</td>
<td>Global timestamping implemented with a 64-bit timestamp value.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b01000.

**Bits [23:18]**

Reserved, RES0.

**TRCEXDATA, bit [17]**

When TRCIDR0.TRCDATA ≠ 0b00:

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>TRCEXDATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing of data transfers for exceptions and exception returns not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing of data transfers for exceptions and exception returns implemented.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

**QSUPP, bits [16:15]**

Indicates that the trace unit implements Q element support.

<table>
<thead>
<tr>
<th>QSUPP</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Q element support is not implemented.</td>
</tr>
<tr>
<td>0b01</td>
<td>Q element support is implemented, and only supports Q elements with instruction counts.</td>
</tr>
<tr>
<td>0b10</td>
<td>Q element support is implemented, and only supports Q elements without instruction counts.</td>
</tr>
<tr>
<td>0b11</td>
<td>Q element support is implemented, and supports:</td>
</tr>
<tr>
<td></td>
<td>• Q elements with instruction counts.</td>
</tr>
<tr>
<td></td>
<td>• Q elements without instruction counts.</td>
</tr>
</tbody>
</table>

**QFILT, bit [14]**

When TRCIDR0.QSUPP ≠ 0b00:

Indicates if the trace unit implements Q element filtering.

<table>
<thead>
<tr>
<th>QFILT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Q element filtering is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Q element filtering is implemented.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.
CONDTYPE, bits [13:12]

When TRCIDR0.TRCCOND == 0b1:

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>CONDTYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Conditional instructions are traced with an indication of whether they pass or fail their condition code check.</td>
</tr>
<tr>
<td>0b01</td>
<td>Conditional instructions are traced with an indication of the APSR condition flags.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == 0b0000:

Indicates the number of ETEEvents implemented.

<table>
<thead>
<tr>
<th>NUMEVENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The trace unit supports 0 ETEEvents.</td>
</tr>
</tbody>
</table>

All other values are reserved.

When TRCIDR4.NUMRSPAIR != 0b0000:

Indicates the number of ETEEvents implemented.

<table>
<thead>
<tr>
<th>NUMEVENT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>The trace unit supports 1 ETEEvent.</td>
</tr>
<tr>
<td>0b01</td>
<td>The trace unit supports 2 ETEEvents.</td>
</tr>
<tr>
<td>0b10</td>
<td>The trace unit supports 3 ETEEvents.</td>
</tr>
<tr>
<td>0b11</td>
<td>The trace unit supports 4 ETEEvents.</td>
</tr>
</tbody>
</table>

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

<table>
<thead>
<tr>
<th>RETSTACK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Return stack not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Return stack implemented.</td>
</tr>
</tbody>
</table>

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.
<table>
<thead>
<tr>
<th>TRCCCI</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Cycle counting not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Cycle counting implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

**TRCOND, bit [6]**

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

<table>
<thead>
<tr>
<th>TRCOND</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Conditional instruction tracing not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Conditional instruction tracing implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

**TRCBB, bit [5]**

Indicates if the trace unit implements branch broadcasting.

<table>
<thead>
<tr>
<th>TRCBB</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Branch broadcasting not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Branch broadcasting implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

**TRCDATA, bits [4:3]**

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

<table>
<thead>
<tr>
<th>TRCDATA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Data tracing not implemented.</td>
</tr>
<tr>
<td>0b11</td>
<td>Data tracing implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

**INSTP0, bits [2:1]**

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

<table>
<thead>
<tr>
<th>INSTP0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Load and store instructions are not P0 instructions.</td>
</tr>
<tr>
<td>0b11</td>
<td>Load and store instructions are P0 instructions.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00.

**Bit [0]**

Reserved, RES1.

**Accessing the TRCIDR0**

TRCIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1E0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.
The TRCIDR1 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR1 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR1[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR1 are RES0.

**Attributes**

TRCIDR1 is a 32-bit register.

**Field descriptions**

The TRCIDR1 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESIGNER</td>
<td>RES0</td>
<td>RES1</td>
<td>TRCARCHMAJ</td>
<td>TRCARCHMIN</td>
<td>REVISION</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DESIGNER, bits [31:24]**

Indicates which company designed the trace unit. The permitted values of this field are the same as MIDR_EL1. Implementer.

**Bits [23:16]**

Reserved, RES0.

**Bits [15:12]**

Reserved, RES1.

**TRCARCHMAJ, bits [11:8]**

Major architecture version.

<table>
<thead>
<tr>
<th>TRCARCHMAJ</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b1111.

**TRCARCHMIN, bits [7:4]**

Minor architecture version.
Meaning

<table>
<thead>
<tr>
<th>TRCARCHMIN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b1111</td>
<td>If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b1111.

**REVISION, bits [3:0]**

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of:

- The trace registers.
- The OS Lock registers.

Arm recommends that the initial implementation sets REVISION == 0x0 and the field then increments for any subsequent implementations. However, it is acceptable to omit some values or use another scheme to identify the revision number.

Arm recommends that TRCPIDR2.REVISION == TRCIDR1.REVISION. However, in situations where it is difficult to align these fields, such as with a metal layer fix then it is acceptable to change the REVISION fields independently.

**Accessing the TRCIDR1**

**TRCIDR1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1E4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR10, ID Register 10

The TRCIDR10 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR10 bits [31:0] are architecturally mapped to AArch64 System register `TRCIDR10[31:0]`.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR10 are `RES0`.

**Attributes**

TRCIDR10 is a 32-bit register.

**Field descriptions**

The TRCIDR10 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| NUMP1KEY, bits [31:0] |

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, `RES0`.

**Accessing the TRCIDR10**

**TRCIDR10 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x188</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.
TRCIDR11, ID Register 11

The TRCIDR11 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR11 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR11[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR11 are RES0.

**Attributes**

TRCIDR11 is a 32-bit register.

**Field descriptions**

The TRCIDR11 bit assignments are:

```
    31  30  29  28  27  26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9  8  7  6  5  4  3  2  1  0
```

<table>
<thead>
<tr>
<th>NUMP1SPC</th>
</tr>
</thead>
</table>

**NUMP1SPC, bits [31:0]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

**Accessing the TRCIDR11**

TRCIDR11 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x18C</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR12, ID Register 12

The TRCIDR12 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR12 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR12[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR12 are RES0.

**Attributes**

TRCIDR12 is a 32-bit register.

**Field descriptions**

The TRCIDR12 bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
```

**NUMCONDKEY, bits [31:0]**

When TRCIDR0.TRCCOND == 0b1:

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

**Accessing the TRCIDR12**

TRCIDR12 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x190</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
**TRCIDR13, ID Register 13**

The TRCIDR13 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR13 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR13[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR13 are RES0.

**Attributes**

TRCIDR13 is a 32-bit register.

**Field descriptions**

The TRCIDR13 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | NUMCONDSPC |

**NUMCONDSPC, bits [31:0]**

When TRCIDR0.TRCCOND == 0b1:

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures. Otherwise:

Reserved, RES0.

**Accessing the TRCIDR13**

**TRCIDR13 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x194</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR2, ID Register 2

The TRCIDR2 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR2 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR2[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR2 are RES0.

Attributes

TRCIDR2 is a 32-bit register.

Field descriptions

The TRCIDR2 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>WFXMODE</td>
<td>VMIDOPT</td>
<td>CCSIZE</td>
<td>DVSIZE</td>
<td>DASIZE</td>
<td>VMIDSIZE</td>
<td>CIDSIZE</td>
<td>IASIZE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

WFXMODE, bit [31]

Indicates whether WFI and WFE instructions are classified as P0 instructions:

<table>
<thead>
<tr>
<th>WFXMODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>WFI and WFE instructions are not classified as P0 instructions.</td>
</tr>
<tr>
<td>0b1</td>
<td>WFI and WFE instructions are classified as P0 instructions.</td>
</tr>
</tbody>
</table>

VMIDOPT, bits [30:29]

When TRCIDR2.VMIDSIZE != 0b00000:

Indicates the options for Virtual context identifier selection.

<table>
<thead>
<tr>
<th>VMIDOPT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES0.</td>
</tr>
<tr>
<td>0b1</td>
<td>Virtual context identifier selection supported. TRCCONFIGR.VMIDOPT is implemented.</td>
</tr>
<tr>
<td>0b10</td>
<td>Virtual context identifier selection not supported. TRCCONFIGR.VMIDOPT is RES1.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b10.

Otherwise:

Reserved, RES0.
**CCSIZE, bits [28:25]**

When TRCIDR0.TRCCCI == 0b1:

Indicates the size of the cycle counter.

<table>
<thead>
<tr>
<th>CCSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The cycle counter is 12 bits in length.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The cycle counter is 13 bits in length.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The cycle counter is 14 bits in length.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The cycle counter is 15 bits in length.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The cycle counter is 16 bits in length.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The cycle counter is 17 bits in length.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The cycle counter is 18 bits in length.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The cycle counter is 19 bits in length.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The cycle counter is 20 bits in length.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**DVSIZE, bits [24:20]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>DVSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Data value tracing not implemented.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Data value tracing has a maximum of 32-bit data values.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Data value tracing has a maximum of 64-bit data values.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.

**DASIZE, bits [19:15]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>DASIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Data address tracing not implemented.</td>
</tr>
<tr>
<td>0b0100</td>
<td>Data address tracing has a maximum of 32-bit data addresses.</td>
</tr>
<tr>
<td>0b1000</td>
<td>Data address tracing has a maximum of 64-bit data addresses.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Otherwise:

Reserved, RES0.
VMIDSIZE, bits [14:10]

When HaveEL(EL2):

Indicates the trace unit Virtual context identifier size.

<table>
<thead>
<tr>
<th>VMIDSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Virtual context identifier tracing is not supported.</td>
</tr>
<tr>
<td>0b00001</td>
<td>8-bit Virtual context identifier size.</td>
</tr>
<tr>
<td>0b00010</td>
<td>16-bit Virtual context identifier size.</td>
</tr>
<tr>
<td>0b00100</td>
<td>32-bit Virtual context identifier size.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00100.

Otherwise:

Reserved, RES0.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

<table>
<thead>
<tr>
<th>CIDSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Context identifier tracing is not supported.</td>
</tr>
<tr>
<td>0b00100</td>
<td>32-bit Context identifier size.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b00100.

IASIZE, bits [4:0]

Virtual instruction address size.

<table>
<thead>
<tr>
<th>IASIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000</td>
<td>Maximum of 32-bit instruction address size.</td>
</tr>
<tr>
<td>0b01000</td>
<td>Maximum of 64-bit instruction address size.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b01000.

Accessing the TRCIDR2

TRCIDR2 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1E8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR3, ID Register 3

The TRCIDR3 characteristics are:

**Purpose**

Returns the base architecture of the trace unit.

**Configuration**

External register TRCIDR3 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR3[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR3 are RES0.

**Attributes**

TRCIDR3 is a 32-bit register.

**Field descriptions**

The TRCIDR3 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOOVERFLOW</td>
<td>NUMPROC[2:0]</td>
<td>SYSSTALL</td>
<td>STALLCTL</td>
<td>SYNCPTR</td>
<td>TRCERR</td>
<td>RES0</td>
<td>EXLEVEL_NS_EL2</td>
<td>EXLEVEL_NS_EL1</td>
<td>EXLEVEL_S_EL3</td>
<td>EXLEVEL_S_EL2</td>
<td>EXLEVEL_S_EL1</td>
</tr>
</tbody>
</table>

**NOOVERFLOW, bit [31]**

Indicates if overflow prevention is implemented.

<table>
<thead>
<tr>
<th>NOOVERFLOW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Overflow prevention is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Overflow prevention is implemented.</td>
</tr>
</tbody>
</table>

**NUMPROC[2:0], bits [30:28]**

This field is bits[2:0] of NUMPROC[4:0].

Indicates the number of PEs available for tracing.

<table>
<thead>
<tr>
<th>NUMPROC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000000</td>
<td>The trace unit can trace one PE.</td>
</tr>
</tbody>
</table>

This field reads as 0b000000.

The NUMPROC field is split as follows:

- NUMPROC[2:0] is TRCIDR3[30:28].
- NUMPROC[4:3] is TRCIDR3[13:12].

**SYSSTALL, bit [27]**

When TRCIDR3.STALLCTL == 0b1:

Indicates if stalling of the PE is permitted.

<table>
<thead>
<tr>
<th>SYSSTALL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stalling of the PE is not permitted.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stalling of the PE is permitted.</td>
</tr>
</tbody>
</table>
The value of this field might be dynamic and change based on system conditions.

Otherwise:

Reserved, RES0.

**STALLCTL, bit [26]**

Indicates if trace unit implements stalling of the PE.

<table>
<thead>
<tr>
<th>STALLCTL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stalling of the PE is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Stalling of the PE is implemented.</td>
</tr>
</tbody>
</table>

**SYNCPR, bit [25]**

Indicates if an implementation has a fixed synchronization period.

<table>
<thead>
<tr>
<th>SYNCPR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>TRCSYNCPR is read-write so software can change the synchronization period.</td>
</tr>
<tr>
<td>0b1</td>
<td>TRCSYNCPR is read-only so the synchronization period is fixed.</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

**TRCERR, bit [24]**

Indicates forced tracing of System Error exceptions is implemented.

<table>
<thead>
<tr>
<th>TRCERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forced tracing of System Error exceptions is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forced tracing of System Error exceptions is implemented.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

**Bit [23]**

Reserved, RES0.

**EXLEVEL_NS_EL2, bit [22]**

Indicates if Non-secure EL2 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL2 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL2 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_NS_EL1, bit [21]**

Indicates if Non-secure EL1 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL1 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL1 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_NS_EL0, bit [20]**

Indicates if Non-secure EL0 implemented.
<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Non-secure EL0 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Non-secure EL0 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL3, bit [19]**
Indicates if Secure EL3 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL3</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL3 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL3 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL2, bit [18]**
Indicates if Secure EL2 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL2 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL2 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL1, bit [17]**
Indicates if Secure EL1 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL1 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL1 is implemented.</td>
</tr>
</tbody>
</table>

**EXLEVEL_S_EL0, bit [16]**
Indicates if Secure EL0 implemented.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Secure EL0 is not implemented.</td>
</tr>
<tr>
<td>0b1</td>
<td>Secure EL0 is implemented.</td>
</tr>
</tbody>
</table>

**Bits [15:14]**
Reserved, RES0.

**NUMPROC[4:3], bits [13:12]**
This field is bits[4:3] of NUMPROC[4:0].
See NUMPROC[2:0] for the field description.

**CCITMIN, bits [11:0]**

When TRCIDR0.TRCCCI == 0b1:
Indicates the minimum value that can be programmed in TRCCCTL.TR.THRESHOLD.
When cycle counting in the instruction trace is supported, that is TRCIDR0.TRCCCI == 0b1, then the minimum value of this field is 0x001.

Otherwise:
Reserved, RES0.
Accessing the TRCIDR3

TRCIDR3 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1EC</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR4, ID Register 4

The TRCIDR4 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR4 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR4[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR4 are RES0.

**Attributes**

TRCIDR4 is a 32-bit register.

**Field descriptions**

The TRCIDR4 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMVMIDC</td>
<td>NUMCIDC</td>
<td>NUMSSCC</td>
<td>NUMRSPAIR</td>
<td>NUMPC</td>
<td>RES0</td>
<td>SUPPDAC</td>
<td>NUMDVC</td>
<td>NUMACPAIRS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NUMVMIDC, bits [31:28]**

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

<table>
<thead>
<tr>
<th>NUMVMIDC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Virtual Context Identifier Comparators are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Virtual Context Identifier Comparator.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Virtual Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Virtual Context Identifier Comparators.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NUMCIDC, bits [27:24]**

Indicates the number of Context Identifier Comparators that are available for tracing.
<table>
<thead>
<tr>
<th>NUMCIDC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Context Identifier Comparators are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Context Identifier Comparator.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Context Identifier Comparators.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Context Identifier Comparators.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NUMSSCC, bits [23:20]**

Indicates the number of Single-shot Comparator Controls that are available for tracing.

<table>
<thead>
<tr>
<th>NUMSSCC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No Single-shot Comparator Controls are available.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The implementation has one Single-shot Comparator Control.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The implementation has two Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The implementation has three Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The implementation has four Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The implementation has five Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The implementation has six Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The implementation has seven Single-shot Comparator Controls.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The implementation has eight Single-shot Comparator Controls.</td>
</tr>
</tbody>
</table>

All other values are reserved.

**NUMRSPAIR, bits [19:16]**

Indicates the number of resource selector pairs that are available for tracing.
### NUMRSPAIR

<table>
<thead>
<tr>
<th>Meaning</th>
<th>( \text{0b}xx )</th>
</tr>
</thead>
<tbody>
<tr>
<td>The implementation has zero resource selectors.</td>
<td>0b0000</td>
</tr>
<tr>
<td>The implementation has two resource selector pairs.</td>
<td>0b0001</td>
</tr>
<tr>
<td>The implementation has three resource selector pairs.</td>
<td>0b0010</td>
</tr>
<tr>
<td>The implementation has four resource selector pairs.</td>
<td>0b0011</td>
</tr>
<tr>
<td>The implementation has five resource selector pairs.</td>
<td>0b0100</td>
</tr>
<tr>
<td>The implementation has six resource selector pairs.</td>
<td>0b0101</td>
</tr>
<tr>
<td>The implementation has seven resource selector pairs.</td>
<td>0b0110</td>
</tr>
<tr>
<td>The implementation has eight resource selector pairs.</td>
<td>0b0111</td>
</tr>
<tr>
<td>The implementation has nine resource selector pairs.</td>
<td>0b1000</td>
</tr>
<tr>
<td>The implementation has ten resource selector pairs.</td>
<td>0b1001</td>
</tr>
<tr>
<td>The implementation has eleven resource selector pairs.</td>
<td>0b1010</td>
</tr>
<tr>
<td>The implementation has twelve resource selector pairs.</td>
<td>0b1011</td>
</tr>
<tr>
<td>The implementation has thirteen resource selector pairs.</td>
<td>0b1100</td>
</tr>
<tr>
<td>The implementation has fourteen resource selector pairs.</td>
<td>0b1101</td>
</tr>
<tr>
<td>The implementation has fifteen resource selector pairs.</td>
<td>0b1110</td>
</tr>
<tr>
<td>The implementation has sixteen resource selector pairs.</td>
<td>0b1111</td>
</tr>
</tbody>
</table>

All other values are reserved.

### NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>( \text{0b}xx )</th>
</tr>
</thead>
<tbody>
<tr>
<td>No PE Comparator Inputs are available.</td>
<td>0b0000</td>
</tr>
<tr>
<td>The implementation has one PE Comparator Input.</td>
<td>0b0001</td>
</tr>
<tr>
<td>The implementation has two PE Comparator Inputs.</td>
<td>0b0010</td>
</tr>
<tr>
<td>The implementation has three PE Comparator Inputs.</td>
<td>0b0011</td>
</tr>
<tr>
<td>The implementation has four PE Comparator Inputs.</td>
<td>0b0100</td>
</tr>
<tr>
<td>The implementation has five PE Comparator Inputs.</td>
<td>0b0101</td>
</tr>
<tr>
<td>The implementation has six PE Comparator Inputs.</td>
<td>0b0110</td>
</tr>
<tr>
<td>The implementation has seven PE Comparator Inputs.</td>
<td>0b0111</td>
</tr>
<tr>
<td>The implementation has eight PE Comparator Inputs.</td>
<td>0b1000</td>
</tr>
</tbody>
</table>

All other values are reserved.

### Bits [11:9]

Reserved, RES0.

### SUPPDAC, bit [8]

**When TRCIDR4.NUMACPAIRS \(!= \text{0b}0000\):**

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>( \text{0b}x )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data address comparisons not implemented.</td>
<td>0b0</td>
</tr>
<tr>
<td>Data address comparisons implemented.</td>
<td>0b1</td>
</tr>
</tbody>
</table>

This bit reads as \( \text{0b}0 \).

**Otherwise:**

Reserved, RES0.

### NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.
### NUMDVC, bits [5:0]

This field reads as 0b0000.

Indicates the number of data value comparators that are available for tracing.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No data value comparators implemented.</td>
<td>0b0000</td>
</tr>
<tr>
<td>One data value comparator implemented.</td>
<td>0b0001</td>
</tr>
<tr>
<td>Two data value comparators implemented.</td>
<td>0b0010</td>
</tr>
<tr>
<td>Three data value comparators implemented.</td>
<td>0b0011</td>
</tr>
<tr>
<td>Four data value comparators implemented.</td>
<td>0b0100</td>
</tr>
<tr>
<td>Five data value comparators implemented.</td>
<td>0b0101</td>
</tr>
<tr>
<td>Six data value comparators implemented.</td>
<td>0b0110</td>
</tr>
<tr>
<td>Seven data value comparators implemented.</td>
<td>0b0111</td>
</tr>
<tr>
<td>Eight data value comparators implemented.</td>
<td>0b1000</td>
</tr>
</tbody>
</table>

All other values are reserved.

### NUMACPAIRS, bits [3:0]

Indicates the number of Address Comparator pairs that are available for tracing.

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>No Address Comparator pairs are available.</td>
<td>0b0000</td>
</tr>
<tr>
<td>The implementation has one Address Comparator pair.</td>
<td>0b0001</td>
</tr>
<tr>
<td>The implementation has two Address Comparator pairs.</td>
<td>0b0010</td>
</tr>
<tr>
<td>The implementation has three Address Comparator pairs.</td>
<td>0b0011</td>
</tr>
<tr>
<td>The implementation has four Address Comparator pairs.</td>
<td>0b0100</td>
</tr>
<tr>
<td>The implementation has five Address Comparator pairs.</td>
<td>0b0101</td>
</tr>
<tr>
<td>The implementation has six Address Comparator pairs.</td>
<td>0b0110</td>
</tr>
<tr>
<td>The implementation has seven Address Comparator pairs.</td>
<td>0b0111</td>
</tr>
<tr>
<td>The implementation has eight Address Comparator pairs.</td>
<td>0b1000</td>
</tr>
</tbody>
</table>

All other values are reserved.

### Accessing the TRCIDR4

**TRCIDR4 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1F0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.
TRCIDR5, ID Register 5

The TRCIDR5 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR5 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR5[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR5 are RES0.

Attributes

TRCIDR5 is a 32-bit register.

Field descriptions

The TRCIDR5 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>NUMCNTR</td>
</tr>
<tr>
<td>29</td>
<td>NUMSEQSTATE</td>
</tr>
<tr>
<td>28</td>
<td>LPOVERRIDE</td>
</tr>
<tr>
<td>27</td>
<td>ATBTRIG</td>
</tr>
<tr>
<td>26</td>
<td>TRACEIDSIZE</td>
</tr>
<tr>
<td>25</td>
<td>RES0</td>
</tr>
<tr>
<td>24</td>
<td>NUMEXTINSEL</td>
</tr>
<tr>
<td>23</td>
<td>NUMEXTIN</td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Bit [31]

Reserved, RES0.

NUMCNTR, bits [30:28]

Indicates the number of Counters that are available for tracing.

<table>
<thead>
<tr>
<th>NUMCNTR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>No Counters are available.</td>
</tr>
<tr>
<td>0b001</td>
<td>One Counter implemented.</td>
</tr>
<tr>
<td>0b010</td>
<td>Two Counters implemented.</td>
</tr>
<tr>
<td>0b011</td>
<td>Three Counters implemented.</td>
</tr>
<tr>
<td>0b100</td>
<td>Four Counters implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.

NUMSEQSTATE, bits [27:25]

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

<table>
<thead>
<tr>
<th>NUMSEQSTATE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>The Sequencer is not implemented.</td>
</tr>
<tr>
<td>0b100</td>
<td>Four Sequencer states are implemented.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.
Bit [24]

Reserved, RES0.

LPOVERRIDE, bit [23]

Indicates support for Low-power Override Mode.

<table>
<thead>
<tr>
<th>LPOVERRIDE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit does not support Low-power Override Mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit support Low-power Override Mode.</td>
</tr>
</tbody>
</table>

ATBTRIG, bit [22]

Indicates if the implementation can support ATB triggers.

<table>
<thead>
<tr>
<th>ATBTRIG</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The implementation does not support ATB triggers.</td>
</tr>
<tr>
<td>0b1</td>
<td>The implementation supports ATB triggers.</td>
</tr>
</tbody>
</table>

This bit is res0 if TRCIDR4.NUMRSPAIR == 0b0000.

TRACEIDSIZE, bits [21:16]

Indicates the trace ID width.

<table>
<thead>
<tr>
<th>TRACEIDSIZE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00000000</td>
<td>The external trace interface is not implemented.</td>
</tr>
<tr>
<td>0b000111</td>
<td>The implementation supports a 7-bit trace ID.</td>
</tr>
</tbody>
</table>

All other values are reserved.

Note that AMBA ATB requires a 7-bit trace ID width.

Bits [15:12]

Reserved, RES0.

NUMEXTINSEL, bits [11:9]

Indicates how many External Input Selector resources are implemented.

<table>
<thead>
<tr>
<th>NUMEXTINSEL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>No External Input Selector resources are available.</td>
</tr>
<tr>
<td>0b01</td>
<td>1 External Input Selector resource is available.</td>
</tr>
<tr>
<td>0b10</td>
<td>2 External Input Selector resources are available.</td>
</tr>
<tr>
<td>0b11</td>
<td>3 External Input Selector resources are available.</td>
</tr>
<tr>
<td>0b100</td>
<td>4 External Input Selector resources are available.</td>
</tr>
</tbody>
</table>

All other values are reserved.

NUMEXTIN, bits [8:0]

Indicates how many External Inputs are implemented.

<table>
<thead>
<tr>
<th>NUMEXTIN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b11111111</td>
<td>Unified PMU event selection.</td>
</tr>
</tbody>
</table>

All other values are reserved.
Accessing the TRCIDR5

TRCIDR5 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1F4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR6, ID Register 6

The TRCIDR6 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR6 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR6[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR6 are RES0.

**Attributes**

TRCIDR6 is a 32-bit register.

**Field descriptions**

The TRCIDR6 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | RES0 |

**Bits [31:0]**

Reserved, RES0.

**Accessing the TRCIDR6**

TRCIDR6 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1F8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR7, ID Register 7

The TRCIDR7 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR7 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR7[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR7 are RES0.

**Attributes**

TRCIDR7 is a 32-bit register.

**Field descriptions**

The TRCIDR7 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0.</td>
</tr>
</tbody>
</table>

**Accessing the TRCIDR7**

TRCIDR7 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1FC</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
The TRCIDR8 characteristics are:

**Purpose**

Returns the maximum speculation depth of the instruction trace element stream.

**Configuration**

External register TRCIDR8 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR8[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR8 are RES0.

**Attributes**

TRCIDR8 is a 32-bit register.

**Field descriptions**

The TRCIDR8 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | MAXSPEC |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**MAXSPEC, bits [31:0]**

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

**Accessing the TRCIDR8**

TRCIDR8 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x180</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIDR9, ID Register 9

The TRCIDR9 characteristics are:

**Purpose**

Returns the tracing capabilities of the trace unit.

**Configuration**

External register TRCIDR9 bits [31:0] are architecturally mapped to AArch64 System register TRCIDR9[31:0]. This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIDR9 are RES0.

**Attributes**

TRCIDR9 is a 32-bit register.

**Field descriptions**

The TRCIDR9 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NUMP0KEY, bits [31:0]**

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

**Accessing the TRCIDR9**

TRCIDR9 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x184</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCIMSPEC0, IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

**Purpose**

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

**Configuration**

External register TRCIMSPEC0 bits [31:0] are architecturally mapped to AArch64 System register TRCIMSPEC0[31:0].

Some or all RW fields of this register have defined reset values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCIMSPEC0 are RES0.

**Attributes**

TRCIMSPEC0 is a 32-bit register.

**Field descriptions**

The TRCIMSPEC0 bit assignments are:

| Bit 31 | Bit 30 | Bit 29 | Bit 28 | Bit 27 | Bit 26 | Bit 25 | Bit 24 | Bit 23 | Bit 22 | Bit 21 | Bit 20 | Bit 19 | Bit 18 | Bit 17 | Bit 16 | Bit 15 | Bit 14 | Bit 13 | Bit 12 | Bit 11 | Bit 10 | Bit 9 | Bit 8 | Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| RES0  | EN    | SUPPORT |

**Bits [31:8]**

Reserved, RES0.

**EN, bits [7:4]**

When TRCIMSPEC0.SUPPORT != 0b0000:

Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>The IMPLEMENTATION DEFINED features are not enabled. The trace unit must behave as if the IMPLEMENTATION DEFINED features are not supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b0010</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b0011</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b0100</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b0101</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b0110</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b0111</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1000</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1001</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1010</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1011</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1100</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1101</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1110</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
<tr>
<td>0b1111</td>
<td>The trace unit behavior is IMPLEMENTATION DEFINED.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to 0.
Otherwise:

Reserved, RES0.

**SUPPORT, bits [3:0]**

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

<table>
<thead>
<tr>
<th>SUPPORT</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>No IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0001</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0010</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0011</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0100</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0101</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0110</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b0111</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1000</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1001</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1010</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1011</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1100</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1101</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1110</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
<tr>
<td>0b1111</td>
<td>IMPLEMENTATION DEFINED features are supported.</td>
</tr>
</tbody>
</table>

Use of nonzero values requires written permission from Arm.

Access to this field is **RO**.

**Accessing the TRCIMSPEC0**

**TRCIMSPEC0 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1C0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
TRCIMSPEC<n>, IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

**Purpose**

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

**Configuration**

External register TRCIMSPEC<n> bits [31:0] are architecturally mapped to AArch64 System register TRCIMSPEC<n>[31:0].

This register is present only when the trace unit implements this OPTIONAL register and ETE is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are RES0.

**Attributes**

TRCIMSPEC<n> is a 32-bit register.

**Field descriptions**

The TRCIMSPEC<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    | | | | |

**IMPLEMENTATION DEFINED, bits [31:0]**

IMPLEMENTATION DEFINED, IMPLEMENTATION_DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

**Accessing the TRCIMSPEC<n>**

TRCIMSPEC<n> can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x1C0 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
The TRCITCTRL characteristics are:

**Purpose**

A component can use TRCITCTRL to dynamically switch between functional mode and integration mode. In integration mode, topology detection is enabled. After switching to integration mode and performing integration tests or topology detection, reset the system to ensure correct behavior of CoreSight and other connected system components.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCITCTRL are **RES0**.

There are no configuration notes.

**Attributes**

TRCITCTRL is a 32-bit register.

**Field descriptions**

The TRCITCTRL bit assignments are:

```
31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
   RES0 IME
```

**Bits [31:1]**

Reserved, **RES0**.

**IME, bit [0]**

Integration Mode Enable.

<table>
<thead>
<tr>
<th>IME</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The component must enter functional mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>The component must enter integration mode, and enable support for topology detection and integration testing.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if no topology detection or integration functionality is implemented.

**Accessing the TRCITCTRL**

External debugger accesses to this register are **IMPLEMENTATION DEFINED** when the trace unit is not in the Idle state.

**TRCITCTRL can be accessed through the external debug interface:**

```
<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xF00</td>
</tr>
</tbody>
</table>
```

This interface is accessible as follows:
• When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
• Otherwise accesses to this register are RW.
TRCLAR, Lock Access Register

The TRCLAR characteristics are:

Purpose

Used to lock and unlock the Software Lock.

Note that ETE does not implement the Software Lock.

For additional information see the CoreSight Architecture Specification.

Configuration

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCLAR are RES0.

There are no configuration notes.

Attributes

TRCLAR is a 32-bit register.

Field descriptions

The TRCLAR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| KEY |

KEY, bits [31:0]

When the Software Lock is implemented:

Software Lock Key.

A value of 0xC5ACCE55 unlocks the Software Lock.

Any other value locks the Software Lock.

Otherwise:

Reserved, RES0.

Accessing the TRCLAR

External debugger accesses to this register are unaffected by the OS Lock.

TRCLAR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFB0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are WO.
TRCLSR, Lock Status Register

The TRCLSR characteristics are:

**Purpose**

Indicates whether the Software Lock is implemented, and the current status of the Software Lock.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCLSR are RES0.

There are no configuration notes.

**Attributes**

TRCLSR is a 32-bit register.

**Field descriptions**

The TRCLSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>30</td>
<td>nTT</td>
<td>Software lock size.</td>
</tr>
<tr>
<td>29</td>
<td>SLK</td>
<td>The current Software Lock status.</td>
</tr>
<tr>
<td>28</td>
<td>0</td>
<td>0b0: Software Lock is unlocked. Writes to the other registers in this component, except for the TRCLAR, are ignored.</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>0b1: Software Lock is locked.</td>
</tr>
<tr>
<td>26</td>
<td>SLI</td>
<td>Indicates whether the Software Lock is implemented.</td>
</tr>
<tr>
<td>25</td>
<td>0</td>
<td>0b0: Software Lock is not implemented. Writes to the TRCLAR are ignored.</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>0b1: Software Lock is implemented.</td>
</tr>
</tbody>
</table>

This bit reads as zero.
This bit reads as zero.

**Accessing the TRCLSR**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCLSR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFB4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.
TRCOSLSR, Trace OS Lock Status Register

The TRCOSLSR characteristics are:

**Purpose**

Returns the status of the Trace OS Lock.

**Configuration**

External register TRCOSLSR bits [31:0] are architecturally mapped to AArch64 System register TRCOSLSR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCOSLSR are RES0.

**Attributes**

TRCOSLSR is a 32-bit register.

**Field descriptions**

The TRCOSLSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>OSLM[2:1]</td>
</tr>
<tr>
<td>29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>OSLK</td>
</tr>
<tr>
<td>27</td>
<td>RES0</td>
</tr>
<tr>
<td>26</td>
<td>OSLM[0]</td>
</tr>
</tbody>
</table>

**Bits [31:5]**

Reserved, RES0.

**OSLM[2:1], bits [4:3]**

This field is bits[2:1] of OSLM[2:0].

OS Lock model.

<table>
<thead>
<tr>
<th>OSLM</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b000</td>
<td>Trace OS Lock is not implemented.</td>
</tr>
<tr>
<td>0b010</td>
<td>Trace OS Lock is implemented.</td>
</tr>
<tr>
<td>0b100</td>
<td>Trace OS Lock is not implemented, and the trace unit is controlled by the PE OS Lock.</td>
</tr>
</tbody>
</table>

All other values are reserved.

This field reads as 0b100.

The OSLM field is split as follows:

- OSLM[2:1] is TRCOSLSR[4:3].
- OSLM[0] is TRCOSLSR[0].

**Bit [2]**

Reserved, RES0.

**OSLK, bit [1]**

OS Lock status.
<table>
<thead>
<tr>
<th>OSLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The OS Lock is unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>The OS Lock is locked.</td>
</tr>
</tbody>
</table>

Note that this field indicates the state of the PE OS Lock.

**OSLM[0], bit [0]**

This field is bit[0] of OSLM[2:0].

See OSLM[2:1] for the field description.

**Accessing the TRCOSLSR**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCOSLSR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x304</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCPDCR, PowerDown Control Register

The TRCPDCR characteristics are:

**Purpose**

Requests the system to provide power to the trace unit.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPDCR are RES0.

There are no configuration notes.

**Attributes**

TRCPDCR is a 32-bit register.

**Field descriptions**

The TRCPDCR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>28</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>6</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>5</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>4</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>3</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>2</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>1</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>0</td>
<td>Reserved, RES0</td>
</tr>
</tbody>
</table>

**PU, bit [3]**

Power Up Request.

<table>
<thead>
<tr>
<th>PU</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The system can remove power from the trace unit core power domain, or requests for power to the trace unit core power domain are implemented outside of the trace unit.</td>
</tr>
<tr>
<td>0b1</td>
<td>The system must provide power to the trace unit core power domain.</td>
</tr>
</tbody>
</table>

This bit is RES0.

**Bits [2:0]**

Reserved, RES0.

**Accessing the TRCPDCR**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPDCR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x310</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
TRCPDSR, PowerDown Status Register

The TRCPDSR characteristics are:

**Purpose**

Indicates the power status of the trace unit.

**Configuration**

Some or all RW fields of this register have defined reset values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPDSR are RES0.

There are no configuration notes.

**Attributes**

TRCPDSR is a 32-bit register.

**Field descriptions**

The TRCPDSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RES0</td>
</tr>
</tbody>
</table>

**Bits [31:6]**

Reserved, RES0.

**OSLK, bit [5]**

OS Lock Status.

<table>
<thead>
<tr>
<th>OSLK</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The OS Lock is unlocked.</td>
</tr>
<tr>
<td>0b1</td>
<td>The OS Lock is locked.</td>
</tr>
</tbody>
</table>

Note that this field indicates the state of the PE OS Lock.

**Bits [4:2]**

Reserved, RES0.

**STICKYPD, bit [1]**

Sticky powerdown status. Indicates whether the trace register state is valid.

<table>
<thead>
<tr>
<th>STICKYPD</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The state of TRCOSLSR and the trace registers are valid.</td>
</tr>
<tr>
<td>0b1</td>
<td>The state of TRCOSLSR and the trace registers might not be valid.</td>
</tr>
</tbody>
</table>

This field is set to 0b1 if the power to the trace unit core power domain is removed and the trace unit register state is not valid.
The STICKYPD field is read-sensitive. On a read of the TRCPDSR, this field is cleared to 0b0 after the register has been read.

On a Trace unit reset, this field resets to 1.

**POWER, bit [0]**

Power Status.

<table>
<thead>
<tr>
<th>POWER</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit core power domain is not powered. All trace unit registers are not accessible and they all return an error response.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit core power domain is powered. Trace unit registers are accessible.</td>
</tr>
</tbody>
</table>

Access to this field is RAO.

### Accessing the TRCPDSR

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPDSR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x314</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCPIDR0, Peripheral Identification Register 0

The TRCPIDR0 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR0 are RES0.

There are no configuration notes.

**Attributes**

TRCPIDR0 is a 32-bit register.

**Field descriptions**

The TRCPIDR0 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**RES0**

**PART_0**

**Bits [31:8]**

Reserved, RES0.

**PART_0, bits [7:0]**

IMPLEMENTATION DEFINED.

Part number, bits [7:0].

The part number is selected by the designer of the component, and is stored in TRCPIDR1 PART_1 and TRCPIDR0 PART_0.

This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the TRCPIDR0**

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFE0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
The TRCPIDR1 characteristics are:

**Purpose**

Provides discovery information for the component.
For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR1 are RES0.
There are no configuration notes.

**Attributes**

TRCPIDR1 is a 32-bit register.

**Field descriptions**

The TRCPIDR1 bit assignments are:

| 31  | 30  | 29  | 28  | 27  | 26  | 25  | 24  | 23  | 22  | 21  | 20  | 19  | 18  | 17  | 16  | 15  | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2   | 1   | 0   |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| RES0|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| DES_0|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
| PART_1|     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |

**Bits [31:8]**

Reserved, RES0.

**DES_0, bits [7:4]**

IMPLEMENTATION DEFINED.

Designer, JEP106 identification code, bits [3:0]. TRCPIDR1.DES_0 and TRCPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note that for a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field reads as an IMPLEMENTATION DEFINED value.

**PART_1, bits [3:0]**

IMPLEMENTATION DEFINED.

Part number, bits [11:8].

This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the TRCPIDR1**

External debugger accesses to this register are unaffected by the OS Lock.
TRCPIDR1 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFE4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCPIDR2, Peripheral Identification Register 2

The TRCPIDR2 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR2 are RES0.

There are no configuration notes.

**Attributes**

TRCPIDR2 is a 32-bit register.

**Field descriptions**

The TRCPIDR2 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | REVISION | JEDEC | DES_1 |

**Bits [31:8]**

Reserved, RES0.

**REVISION, bits [7:4]**

IMPLEMENTATION DEFINED.

Component major revision. TRCPIDR2.REVISION and TRCPIDR3.REVAND together form the revision number of the component, with REVISION being the most significant part and REVAND the least significant part. When a component is changed, REVISION or REVAND must be increased to ensure that software can differentiate the different revisions of the component. If REVISION is increased then REVAND should be set to 0.

This field reads as an IMPLEMENTATION DEFINED value.

**JEDEC, bit [3]**

JEDEC-assigned JEP106 implementer code is used.

This bit reads as one.

**DES_1, bits [2:0]**

IMPLEMENTATION DEFINED.

Designer, JEP106 identification code, bits [6:4]. TRCPIDR1.DES_0 and TRCPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.
Note that for a component designed by Arm Limited, the JEP106 identification code is 0x3B. This field reads as an IMPLEMENTATION DEFINED value.

## Accessing the TRCPIDR2

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPIDR2 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFE8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.
TRCPIDR3, Peripheral Identification Register 3

The TRCPIDR3 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR3 are 
RES0.

There are no configuration notes.

**Attributes**

TRCPIDR3 is a 32-bit register.

**Field descriptions**

The TRCPIDR3 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

**RES0**

Reserved, RES0.

**REVAND, bits [7:4]**

IMPLEMENTATION DEFINED.

Component minor revision. TRCPIDR2.REVISION and TRCPIDR3.REVAND together form the revision number of the component, with REVISION being the most significant part and REVAND the least significant part. When a component is changed, REVISION or REVAND must be increased to ensure that software can differentiate the different revisions of the component. If REVISION is increased then REVAND should be set to 0.

This field reads as an IMPLEMENTATION DEFINED value.

**CMOD, bits [3:0]**

IMPLEMENTATION DEFINED.

Customer Modified.

Indicates the component has been modified.

A value of 0b0000 means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field reads as an IMPLEMENTATION DEFINED value.

Accessing the TRCPIDR3

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR3 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>$0xFEC</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCPIDR4, Peripheral Identification Register 4

The TRCPIDR4 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR4 are RES0.

There are no configuration notes.

**Attributes**

TRCPIDR4 is a 32-bit register.

**Field descriptions**

The TRCPIDR4 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | SIZE | DES_2 |

**Bits [31:8]**

Reserved, RES0.

**SIZE, bits [7:4]**

IMPLEMENTATION DEFINED.

Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of 0b0000 means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies \(2^{\text{SIZE}}\) 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This field reads as 0b0000.

**DES_2, bits [3:0]**

IMPLEMENTATION DEFINED.

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be not the same as the
implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC http://www.jedec.org.

Note that for a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value 0x4. This field reads as an IMPLEMENTATION DEFINED value.

**Accessing the TRCPIDR4**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPIDR4 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFD0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
The TRCPIDR5 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR5 are RES0.

There are no configuration notes.

**Attributes**

TRCPIDR5 is a 32-bit register.

**Field descriptions**

The TRCPIDR5 bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:0]**

Reserved, RES0.

**Accessing the TRCPIDR5**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPIDR5 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFD4</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
**TRCPIDR6, Peripheral Identification Register 6**

The TRCPIDR6 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR6 are RES0.

There are no configuration notes.

**Attributes**

TRCPIDR6 is a 32-bit register.

**Field descriptions**

The TRCPIDR6 bit assignments are:

<table>
<thead>
<tr>
<th>Bits [31:0]</th>
<th>RES0</th>
</tr>
</thead>
</table>

Reserved, RES0.

**Accessing the TRCPIDR6**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPIDR6 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFD8</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCPIDR7, Peripheral Identification Register 7

The TRCPIDR7 characteristics are:

**Purpose**

Provides discovery information for the component.

For additional information see the CoreSight Architecture Specification.

**Configuration**

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPIDR7 are RES0.

There are no configuration notes.

**Attributes**

TRCPIDR7 is a 32-bit register.

**Field descriptions**

The TRCPIDR7 bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 |

**Bits [31:0]**

Reserved, RES0.

**Accessing the TRCPIDR7**

External debugger accesses to this register are unaffected by the OS Lock.

**TRCPIDR7 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0xFD0</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RO.
TRCPRGCTRLR, Programming Control Register

The TRCPRGCTRLR characteristics are:

Purpose

Enables the trace unit.

Configuration

External register TRCPRGCTRLR bits [31:0] are architecturally mapped to AArch64 System register TRCPRGCTRLR[31:0].

Some or all RW fields of this register have defined reset values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCPRGCTRLR are RES0.

Attributes

TRCPRGCTRLR is a 32-bit register.

Field descriptions

The TRCPRGCTRLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>29</td>
<td>EN, bit [0]</td>
</tr>
<tr>
<td>28</td>
<td>Trace unit enable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EN</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to 0.

Accessing the TRCPRGCTRLR

Must be programmed.

TRCPRGCTRLR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x004</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCQCTLR, Q Element Control Register

The TRCQCTLR characteristics are:

Purpose

Controls when Q elements are enabled.

Configuration

External register TRCQCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCQCTLR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR0.QFILT == 0b1. Otherwise, direct accesses to TRCQCTLR are RES0.

Attributes

TRCQCTLR is a 32-bit register.

Field descriptions

The TRCQCTLR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved,RES0</td>
</tr>
<tr>
<td>30</td>
<td>MODE</td>
</tr>
<tr>
<td>29</td>
<td>RANGE&lt;m&gt;, bit [m]</td>
</tr>
</tbody>
</table>

Bits [31:9]

Reserved, RES0.

MODE, bit [8]

Selects whether the Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

<table>
<thead>
<tr>
<th>MODE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Exclude mode. The Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map.</td>
</tr>
<tr>
<td>0b1</td>
<td>Include Mode. The Address Range Comparators selected by the RANGE field indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0b0 then Q elements are disabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE<m>, bit [m], for m = 0 to 7

Specifies the Address Range Comparators to be used for controlling Q elements.
<table>
<thead>
<tr>
<th>RANGE&lt;\textit{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\texttt{b0}</td>
<td>The address range that Address Range Comparator \textit{m} defines, is not selected.</td>
</tr>
<tr>
<td>0\texttt{b1}</td>
<td>The address range that Address Range Comparator \textit{m} defines, is selected.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \textit{m} \geq \texttt{TRCIDR4.NUMACPAIRS}.

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the TRCQCTRL**

Must be programmed if \texttt{TRCCONFIGR. QE} \neq \texttt{0b00}.

Writes are \texttt{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

**TRCQCTRL can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x044</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When \texttt{!AllowExternalTraceAccess()}, or \texttt{OSLockStatus()} or \texttt{!IsTraceCorePowered()} accesses to this register generate an error response.
- Otherwise accesses to this register are \texttt{RW}.
The TRCRSCTRL<n> characteristics are:

**Purpose**

Controls the selection of the resources in the trace unit.

**Configuration**

External register TRCRSCTRL<n> bits [31:0] are architecturally mapped to AArch64 System register TRCRSCTRL<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and ( ( ext-TRCIDR4.NUMRSPAIR + 1 ) * 2 ) > n. Otherwise, direct accesses to TRCRSCTRL<n> are RES0.

Resource selector 0 always returns FALSE.

Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

**Attributes**

TRCRSCTRL<n> is a 32-bit register.

**Field descriptions**

The TRCRSCTRL<n> bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | PAIRINV | INV | GROUP | SELECT |

**Bits [31:22]**

Reserved, RES0.

**PAIRINV, bit [21]**

For TRCRSCTRL<n>, where n is even, controls whether the combined result from a resource selector pair is inverted.

<table>
<thead>
<tr>
<th>PAIRINV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Do not invert the combined output of the 2 resource selectors.</td>
</tr>
<tr>
<td>0b1</td>
<td>Invert the combined output of the 2 resource selectors.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**INV, bit [20]**

Controls whether the resource, that GROUP and SELECT selects, is inverted.
**INV**

<table>
<thead>
<tr>
<th><strong>Meaning</strong></th>
<th><strong>INV</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Do not invert the output of this selector.</td>
<td>0b0</td>
</tr>
<tr>
<td>Invert the output of this selector.</td>
<td>0b1</td>
</tr>
</tbody>
</table>

If:

- A is the register TRCRSCTLR<m> where m is even.
- B is the register TRCRSCTLR<m+1>.

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- 0b000 -> A and B.
- 0b001 -> RESERVED.
- 0b010 -> not(A) and B.
- 0b011 -> not(A) and not(B).
- 0b100 -> not(A) or not(B).
- 0b101 -> not(A) or B.
- 0b110 -> RESERVED.
- 0b111 -> A or B.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**GROUP, bits [19:16]**

Selects a group of resources.

<table>
<thead>
<tr>
<th><strong>GROUP</strong></th>
<th><strong>Meaning</strong></th>
<th><strong>SELECT</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>External Input Selectors.</td>
<td>SELECT encoding for External Input Selectors</td>
</tr>
<tr>
<td>0b0001</td>
<td>PE Comparator Inputs.</td>
<td>SELECT encoding for PE Comparator Inputs</td>
</tr>
<tr>
<td>0b0010</td>
<td>Counters and Sequencer.</td>
<td>SELECT encoding for Counters and Sequencer</td>
</tr>
<tr>
<td>0b0011</td>
<td>Single-shot Comparator Controls.</td>
<td>SELECT encoding for Single Comparator Controls</td>
</tr>
<tr>
<td>0b0100</td>
<td>Single Address Comparators.</td>
<td>SELECT encoding for Single Address Comparators</td>
</tr>
<tr>
<td>0b0101</td>
<td>Address Range Comparators.</td>
<td>SELECT encoding for Address Range Comparators</td>
</tr>
<tr>
<td>0b0110</td>
<td>Context Identifier Comparators.</td>
<td>SELECT encoding for Context Identifier Comparators</td>
</tr>
<tr>
<td>0b0111</td>
<td>Virtual Context Identifier Comparators.</td>
<td>SELECT encoding for Virtual Context Identifier Comparators</td>
</tr>
</tbody>
</table>

All other values are reserved.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**SELECT, bits [15:0]**

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.

**SELECT encoding for External Input Selectors**

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>EXTIN&lt;m&gt;, bit [m]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Bits [15:4]**

Reserved, RES0.
EXTIN\(<m\), bit \(m\), for \(m = 0\) to \(3\)

Selects one or more External Inputs.

<table>
<thead>
<tr>
<th>EXTIN(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore EXTIN (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select EXTIN (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for PE Comparator Inputs

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>PECOMP(&lt;m&gt;, bit (m)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [15:8]

Reserved, RES0.

PECOMP\(<m>, bit \(m\), for \(m = 0\) to \(7\)

Selects one or more PE Comparator Inputs.

<table>
<thead>
<tr>
<th>PECOMP(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore PE Comparator Input (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select PE Comparator Input (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Counters and Sequencer

<table>
<thead>
<tr>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>SEQUENCER(&lt;m&gt;, bit ([m+4])</td>
<td>COUNTERS(&lt;m&gt;, bit ([m])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bits [15:8]

Reserved, RES0.

SEQUENCER\(<m>, bit \([m+4]\), for \(m = 0\) to \(3\)

Sequencer states.

<table>
<thead>
<tr>
<th>SEQUENCER(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Sequencer state (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Sequencer state (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS\(<m>, bit \([m]\), for \(m = 0\) to \(3\)

Counters resources at zero.

<table>
<thead>
<tr>
<th>COUNTERS(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Counter (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Counter (m) is zero.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single-shot Comparator Controls
Bits [15:8]

Reserved, RES0.

**SINGLE_SHOT<\textit{m}>, bit [\textit{m}], for \textit{m} = 0 to 7**

Selects one or more Single-shot Comparator Controls.

<table>
<thead>
<tr>
<th>SINGLE_SHOT&lt;\textit{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Single-shot Comparator Control \textit{m}.</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Single-shot Comparator Control \textit{m}.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally \textit{UNKNOWN} value.

**SELECT encoding for Single Address Comparators**

<table>
<thead>
<tr>
<th>Bits [15:8]</th>
<th>SAC&lt;\textit{m}&gt;, bit [\textit{m}]</th>
</tr>
</thead>
</table>

**SAC<\textit{m}>, bit [\textit{m}], for \textit{m} = 0 to 15**

Selects one or more Single Address Comparators.

<table>
<thead>
<tr>
<th>SAC&lt;\textit{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Single Address Comparator \textit{m}.</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Single Address Comparator \textit{m}.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally \textit{UNKNOWN} value.

**SELECT encoding for Address Range Comparators**

<table>
<thead>
<tr>
<th>Bits [15:8]</th>
<th>ARC&lt;\textit{m}&gt;, bit [\textit{m}]</th>
</tr>
</thead>
</table>

**ARC<\textit{m}>, bit [\textit{m}], for \textit{m} = 0 to 7**

Selects one or more Address Range Comparators.

<table>
<thead>
<tr>
<th>ARC&lt;\textit{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Address Range Comparator \textit{m}.</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Address Range Comparator \textit{m}.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally \textit{UNKNOWN} value.

**SELECT encoding for Context Identifier Comparators**

<table>
<thead>
<tr>
<th>Bits [15:8]</th>
<th>CID&lt;\textit{m}&gt;, bit [\textit{m}]</th>
</tr>
</thead>
</table>

Reserved, RES0.
CID\(m\), bit [m], for \(m = 0\) to \(7\)

Selects one or more Context Identifier Comparators.

<table>
<thead>
<tr>
<th>CID(m)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Context Identifier Comparator (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Context Identifier Comparator (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**SELECT encoding for Virtual Context Identifier Comparators**

<table>
<thead>
<tr>
<th></th>
<th>VMID(m), bit [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>RES0</td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [15:8]**

Reserved, RES0.

VMID\(m\), bit [m], for \(m = 0\) to \(7\)

Selects one or more Virtual Context Identifier Comparators.

<table>
<thead>
<tr>
<th>VMID(m)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Ignore Virtual Context Identifier Comparator (m).</td>
</tr>
<tr>
<td>0b1</td>
<td>Select Virtual Context Identifier Comparator (m).</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

**Accessing the TRCRSCTLR\(<n>\)**

Must be programmed if any of the following are true:

- TRCCNTCTRL\(<a>\).RLDEVENT.TYPE == 0b0 and TRCCNTCTRL\(<a>\).RLDEVENT.SEL == n.
- TRCCNTCTRL\(<a>\).RLDEVENT.TYPE == 0b1 and TRCCNTCTRL\(<a>\).RLDEVENT.SEL == n/2.
- TRCCNTCTRL\(<a>\).CNTEVENT.TYPE == 0b0 and TRCCNTCTRL\(<a>\).CNTEVENT.SEL == n.
- TRCCNTCTRL\(<a>\).CNTEVENT.TYPE == 0b1 and TRCCNTCTRL\(<a>\).CNTEVENT.SEL == n/2.
- TRCEVENTCTL0R.EVENT0.TYPE == 0b0 and TRCEVENTCTL0R.EVENT0.SEL == n.
- TRCEVENTCTL0R.EVENT0.TYPE == 0b1 and TRCEVENTCTL0R.EVENT0.SEL == n/2.
- TRCEVENTCTL0R.EVENT1.TYPE == 0b0 and TRCEVENTCTL0R.EVENT1.SEL == n.
- TRCEVENTCTL0R.EVENT1.TYPE == 0b1 and TRCEVENTCTL0R.EVENT1.SEL == n/2.
- TRCEVENTCTL0R.EVENT2.TYPE == 0b0 and TRCEVENTCTL0R.EVENT2.SEL == n.
- TRCEVENTCTL0R.EVENT2.TYPE == 0b1 and TRCEVENTCTL0R.EVENT2.SEL == n/2.
- TRCEVENTCTL0R.EVENT3.TYPE == 0b0 and TRCEVENTCTL0R.EVENT3.SEL == n.
- TRCEVENTCTL0R.EVENT3.TYPE == 0b1 and TRCEVENTCTL0R.EVENT3.SEL == n/2.
- TRCSEQEV\(<a>\).B.TYPE == 0b0 and TRCSEQEVR\(<a>\).B.SEL = n.
- TRCSEQEV\(<a>\).B.TYPE == 0b1 and TRCSEQEVR\(<a>\).B.SEL = n/2.
- TRCSEQEV\(<a>\).F.TYPE == 0b0 and TRCSEQEVR\(<a>\).F.SEL = n.
- TRCSEQEV\(<a>\).F.TYPE == 0b1 and TRCSEQEVR\(<a>\).F.SEL = n/2.
- TRCSEQRSTCTL0R.RST.TYPE == 0b0 and TRCSEQRSTEV\(<a>\).RST.SEL == n.
• **TRCSEQRSTEV**.RST.TYPE == 0b1 and **TRCSEQRSTEV**.RST.SEL == n/2.
• **TRCTSCTLR**.EVENT.TYPE == 0b0 and **TRCTSCTLR**.EVENT.SEL == n.
• **TRCTSCTLR**.EVENT.TYPE == 0b1 and **TRCTSCTLR**.EVENT.SEL == n/2.
• **TRCVICTLR**.EVENT.TYPE == 0b0 and **TRCVICTLR**.EVENT.SEL == n.
• **TRCVICTLR**.EVENT.TYPE == 0b1 and **TRCVICTLR**.EVENT.SEL == n/2.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

****TRCRSCTLR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x200 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

• When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
• Otherwise accesses to this register are **RW**.
The TRCRSR characteristics are:

**Purpose**

Use this to set, or read, the status of the resources.

**Configuration**

External register TRCRSR bits [31:0] are architecturally mapped to AArch64 System register TRCRSR[31:0]. RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCRSR are RES0.

**Attributes**

TRCRSR is a 32-bit register.

**Field descriptions**

The TRCRSR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>TA</td>
</tr>
<tr>
<td>29</td>
<td>EVENT&lt;m&gt;, bit [m+8]</td>
</tr>
<tr>
<td>28</td>
<td>RES0</td>
</tr>
<tr>
<td>27</td>
<td>EXTIN&lt;m&gt;, bit [m]</td>
</tr>
</tbody>
</table>

**Bits [31:13]**

Reserved, RES0.

**TA, bit [12]**

Tracing active.

<table>
<thead>
<tr>
<th>TA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Tracing is not active.</td>
</tr>
<tr>
<td>0b1</td>
<td>Tracing is active.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**EVENT<m>, bit [m+8], for m = 0 to 3**

Untraced status of ETEEEvents.

<table>
<thead>
<tr>
<th>EVENT&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An ETEEEvent[n] has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>An ETEEEvent[n] has occurred while the resources were in the Paused state.</td>
</tr>
</tbody>
</table>

This bit is RES0 if TRCIDR4.NUMRSPAIR == 0 || m > TRCIDR0.NUMEVENT.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Reserved, RES0.

EXTIN<m>, bit [m], for m = 0 to 3

The sticky status of the External Input Selectors.

<table>
<thead>
<tr>
<th>EXTIN&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>An event selected by External Input Selector[n] has not occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>At least one event selected by External Input Selector[n] has occurred while the resources were in the Paused state.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR5.NUMEXTINSEL.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCRSR

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCRSR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x028</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCSEQEVR<n>, Sequencer State Transition Control Register <n>, n = 0 - 2

The TRCSEQEVR<n> characteristics are:

**Purpose**

Moves the Sequencer state:

- Backwards, from state n+1 to state n when a programmed resource event occurs.
- Forwards, from state n to state n+1 when a programmed resource event occurs.

**Configuration**

External register TRCSEQEVR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSEQEVR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQEVR<n> are RES0.

**Attributes**

TRCSEQEVR<n> is a 32-bit register.

**Field descriptions**

The TRCSEQEVR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>B_TYPE[RES0]</td>
</tr>
<tr>
<td>29</td>
<td>B_SEL</td>
</tr>
<tr>
<td>28</td>
<td>F_TYPE[RES0]</td>
</tr>
<tr>
<td>27</td>
<td>F_SEL</td>
</tr>
<tr>
<td>26</td>
<td>Bits [31:16] Reserved, RES0.</td>
</tr>
</tbody>
</table>

**B_TYPE, bit [15]**

Chooses the type of Resource Selector.

Backward field. Defines whether the backward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == 0x14 then when event 0x14 occurs, the Sequencer moves from state 3 to state 2.

<table>
<thead>
<tr>
<th>B_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCSEQEVR&lt;n&gt;.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCSEQEVR&lt;n&gt;.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR&lt;n&gt;.B.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
**Bits [14:13]**

Reserved, RES0.

**B_SEL, bits [12:8]**

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.B.TYPE controls whether TRCSEQEVR<n>.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Backward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**F_TYPE, bit [7]**

Chooses the type of Resource Selector.

Backward field. Defines whether the forward resource event is a single Resource Selector or a Resource Selector pair.

When the resource event occurs then the Sequencer state moves from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == 0x12 then when event 0x12 occurs, the Sequencer moves from state 1 to state 2.

<table>
<thead>
<tr>
<th>F_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCSEQEVR&lt;n&gt;.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCSEQEVR&lt;n&gt;.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR&lt;n&gt;.F.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [6:5]**

Reserved, RES0.

**F_SEL, bits [4:0]**

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.F.TYPE controls whether TRCSEQEVR<n>.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Forward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCSEQEVR<n>**

Must be programmed if TRCRSCTLR<a>.GROUP == 0b0010 and TRCRSCTLR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCSEQEVR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x100 + 4n</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When `!AllowExternalTraceAccess()`, or `OSLockStatus()` or `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
The TRCSEQRSTEV register characteristics are:

**Purpose**

Moves the Sequencer to state 0 when a programmed resource event occurs.

**Configuration**

External register TRCSEQRSTEV bits [31:0] are architecturally mapped to AArch64 System register TRCSEQRSTEV[31:0].

RW fields in this register reset to architecturally unknown values.

This register is present only when ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQRSTEV are 0.

**Attributes**

TRCSEQRSTEV is a 32-bit register.

**Field descriptions**

The TRCSEQRSTEV bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>7</td>
<td>RST_TYPE</td>
<td>Chooses the type of Resource Selector.</td>
</tr>
<tr>
<td>6:5</td>
<td>Reserved</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>4:0</td>
<td>RST_SEL</td>
<td>Defines the selected Resource Selector or pair of Resource Selectors.</td>
</tr>
</tbody>
</table>

- **RST_TYPE, bit [7]**
  - Choose the type of Resource Selector.
  - 
    - **0b0**: A single Resource Selector.
      - TRCSEQRSTEV.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.
    - **0b1**: A Boolean-combined pair of Resource Selectors.
      - TRCSEQRSTEV.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEV.RST.SEL[4] is RES0.

  On a Trace unit reset, this field resets to an architecturally unknown value.

- **RST_SEL, bits [4:0]**
  - Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEV.RST.TYPE controls whether TRCSEQRSTEV.RST_SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.
If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCSEQRSTEV

Must be programmed if TCRSCTLRI<GROUP == 0b0010 and TCRSCTLRI<SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSEQRSTEV can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x118</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCSEQSTR, Sequencer State Register

The TRCSEQSTR characteristics are:

**Purpose**

Use this to set, or read, the Sequencer state.

**Configuration**

External register TRCSEQSTR bits [31:0] are architecturally mapped to AArch64 System register TRCSEQSTR[31:0]. RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQSTR are RES0.

**Attributes**

TRCSEQSTR is a 32-bit register.

**Field descriptions**

The TRCSEQSTR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31-2</td>
<td>Reserved, RES0.</td>
</tr>
<tr>
<td>1-0</td>
<td>STATE</td>
</tr>
</tbody>
</table>

- **STATE, bits [1:0]**
  - Set or returns the state of the Sequencer.
  - 0b00: State 0.
  - 0b01: State 1.
  - 0b10: State 2.
  - 0b11: State 3.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCSEQSTR**

Must be programmed if TRCRSCtrlR<a>.GROUP == 0b0010 and TRCRSCtrlR<a>.SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSEQSTR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x11C</td>
</tr>
</tbody>
</table>
This interface is accessible as follows:

- When `!AllowExternalTraceAccess()`, or `OSLockStatus()` or `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
TRCSSCCR<\n>, Single-shot Comparator Control Register \n, \n = 0 - 7

The TRCSSCCR<\n> characteristics are:

**Purpose**

Controls the corresponding Single-shot Comparator Control resource.

**Configuration**

External register TRCSSCCR<\n> bits [31:0] are architecturally mapped to AArch64 System register TRCSSCCR<\n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMSSCC > \n. Otherwise, direct accesses to TRCSSCCR<\n> are RES0.

**Attributes**

TRCSSCCR<\n> is a 32-bit register.

**Field descriptions**

The TRCSSCCR<\n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>30</td>
<td>RST</td>
<td>Selects the Single-shot Comparator Control mode.</td>
</tr>
<tr>
<td>29</td>
<td>ARC&lt;m&gt;, bit [m+16]</td>
<td>Selects one or more Address Range Comparators for Single-shot control.</td>
</tr>
<tr>
<td>28</td>
<td>SAC&lt;m&gt;, bit [m]</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:25]**

Reserved, RES0.

**RST, bit [24]**

Selects the Single-shot Comparator Control mode.

<table>
<thead>
<tr>
<th>RST</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single-shot Comparator Control is in single-shot mode.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single-shot Comparator Control is in multi-shot mode.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**ARC<m>, bit [m+16], for \m = 0 to 7**

Selects one or more Address Range Comparators for Single-shot control.

<table>
<thead>
<tr>
<th>ARC&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Address Range Comparator \m is not selected for Single-shot control.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Address Range Comparator \m is selected for Single-shot control.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \m >= TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
SAC<\textit{m}>, \textit{bit} \ [\textit{m}], \text{ for } \textit{m} = 0 \text{ to } 15

Selects one or more Single Address Comparators for Single-shot control.

<table>
<thead>
<tr>
<th>SAC&lt;\textit{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single Address Comparator \textit{m}, is not selected for Single-shot control.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single Address Comparator \textit{m}, is selected for Single-shot control.</td>
</tr>
</tbody>
</table>

This bit is \textit{RES}0 if \textit{m} >= 2 \times \text{TRCIDR4.NUMACPAIRS}.

On a Trace unit reset, this field resets to an architecturally \textit{UNKNOWN} value.

**Accessing the TRCSSCCR<\textit{n}>**

Must be programmed if any TRCRSCTL<\textit{a}>.GROUP == 0b0011 and TRCRSCTL<\textit{a}>.SINGLE_SHOT<\textit{n}> == 0b1.

Writes are \textit{CONSTRAINED UNPREDICTABLE} if the trace unit is not in the Idle state.

**TRCSSCCR<\textit{n}> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x280 + 4\textit{n}</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are \textit{RW}.  

Copyright © 2010-2019 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.
TRCSSCSR<n>, Single-shot Comparator Control Status Register <n>, n = 0 - 7

The TRCSSCSR<n> characteristics are:

**Purpose**

Returns the status of the corresponding Single-shot Comparator Control.

**Configuration**

External register TRCSSCSR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSSCSR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCSR<n> are RES0.

**Attributes**

TRCSSCSR<n> is a 32-bit register.

**Field descriptions**

The TRCSSCSR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>STATUS</td>
</tr>
<tr>
<td>30</td>
<td>PENDING</td>
</tr>
<tr>
<td>29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>PC</td>
</tr>
<tr>
<td>27</td>
<td>DV</td>
</tr>
<tr>
<td>26</td>
<td>DA</td>
</tr>
<tr>
<td>25</td>
<td>INST</td>
</tr>
</tbody>
</table>

**STATUS, bit [31]**

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by TRCSSCCR<n>.ARC, TRCSSCCR<n>.SAC, and TRCSSPCICR<n>.PC.

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No match has occurred. When the first match occurs, this field takes a value of 0b1. It remains at 0b1 until explicitly modified by a write to this register.</td>
</tr>
<tr>
<td>0b1</td>
<td>One or more matches has occurred. If TRCSSCCR&lt;n&gt;.RST == 0b0 then:</td>
</tr>
<tr>
<td></td>
<td>• There is only one match and no more matches are possible.</td>
</tr>
<tr>
<td></td>
<td>• Software must reset this bit to 0b0 to re-enable the Single-shot Comparator Control.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**PENDING, bit [30]**

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.

<table>
<thead>
<tr>
<th>PENDING</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>No match has occurred.</td>
</tr>
<tr>
<td>0b1</td>
<td>One or more matches has occurred.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
**Bits [29:4]**

Reserved, RES0.

**PC, bit [3]**

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

<table>
<thead>
<tr>
<th>PC</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This Single-shot Comparator Control does not support PE Comparator Inputs. Selecting any PE Comparator Inputs using the associated TRCSSPCICR&lt;n&gt; results in CONSTRAINED UNPREDICTABLE behavior of the Single-shot Comparator Control resource. The Single-shot Comparator Control might match unexpectedly or might not match.</td>
</tr>
<tr>
<td>0b1</td>
<td>This Single-shot Comparator Control supports PE Comparator Inputs.</td>
</tr>
</tbody>
</table>

Access to this field is **RO**.

**DV, bit [2]**

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>DV</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This Single-shot Comparator Control does not support data value comparisons.</td>
</tr>
<tr>
<td>0b1</td>
<td>This Single-shot Comparator Control supports data value comparisons.</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

Access to this field is **RO**.

**DA, bit [1]**

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

<table>
<thead>
<tr>
<th>DA</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This Single-shot Comparator Control does not support data address comparisons.</td>
</tr>
<tr>
<td>0b1</td>
<td>This Single-shot Comparator Control supports data address comparisons.</td>
</tr>
</tbody>
</table>

This bit reads as 0b0.

Access to this field is **RO**.

**INST, bit [0]**

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.

<table>
<thead>
<tr>
<th>INST</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>This Single-shot Comparator Control does not support instruction address comparisons.</td>
</tr>
<tr>
<td>0b1</td>
<td>This Single-shot Comparator Control supports instruction address comparisons.</td>
</tr>
</tbody>
</table>

This bit reads as 0b1.

Access to this field is **RO**.
**Accessing the TRCSSCSR<n>**

Must be programmed if TRCRSCTRL<n>.GROUP == 0b0011 and TRCRSCTRL<n>.SINGLE_SHOT[n] == 0b1.

Writes are **CONSTRAINED UNPREDICTABLE** if the trace unit is not in the Idle state.

Reads from this register might return an **UNKNOWN** value if the trace unit is not in either of the Idle or Stable states.

**TRCSSCSR<n> can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x2A0 + 4n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
The TRCSSPCICR<n> characteristics are:

**Purpose**

Returns the status of the corresponding Single-shot Comparator Control.

**Configuration**

External register TRCSSPCICR<n> bits [31:0] are architecturally mapped to AArch64 System register TRCSSPCICR<n>[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented, TRCIDR4.NUMSSCC > n, TRCIDR4.NUMPC > 0b0000 and TRCSSCSR<n>.PC == 0b1. Otherwise, direct accesses to TRCSSPCICR<n> are RES0.

**Attributes**

TRCSSPCICR<n> is a 32-bit register.

**Field descriptions**

The TRCSSPCICR<n> bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>PC&lt;0&gt;, bit [0]</td>
</tr>
<tr>
<td>29</td>
<td>PC&lt;1&gt;, bit [1]</td>
</tr>
<tr>
<td>28</td>
<td>PC&lt;2&gt;, bit [2]</td>
</tr>
<tr>
<td>27</td>
<td>PC&lt;3&gt;, bit [3]</td>
</tr>
<tr>
<td>26</td>
<td>PC&lt;4&gt;, bit [4]</td>
</tr>
<tr>
<td>25</td>
<td>PC&lt;5&gt;, bit [5]</td>
</tr>
<tr>
<td>24</td>
<td>PC&lt;6&gt;, bit [6]</td>
</tr>
<tr>
<td>23</td>
<td>PC&lt;7&gt;, bit [7]</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**PC<0>, bit [0], for m = 0 to 7**

Selects one or more PE Comparator Inputs for Single-shot control.

<table>
<thead>
<tr>
<th>PC&lt;0&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The single PE Comparator Input m, is not selected as for Single-shot control.</td>
</tr>
<tr>
<td>0b1</td>
<td>The single PE Comparator Input m, is selected as for Single-shot control.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR4.NUMPC.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCSSPCICR<n>**

Must be programmed if implemented and any TRCRSCTRL<a>.GROUP == 0b0011 and TRCRSCTRL<a>.SINGLE_SHOT[n] == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.
TRCSSPCICR<\n>, Single-shot Processing Element Comparator Input Control Register <\n>, \n = 0 - 7

TRCSSPCICR<\n> can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x2C0 + 4\n</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCSTALLCTRL, Stall Control Register

The TRCSTALLCTRL characteristics are:

**Purpose**

Enables trace unit functionality that prevents trace unit buffer overflows.

**Configuration**

External register TRCSTALLCTRL bits [31:0] are architecturally mapped to AArch64 System register TRCSTALLCTRL[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR3.STALLCTL == 0b1. Otherwise, direct accesses to TRCSTALLCTRL are RES0.

**Attributes**

TRCSTALLCTRL is a 32-bit register.

**Field descriptions**

The TRCSTALLCTRL bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | NOOVERFLOW | RES0 | [STALL] | RES0 | LEVEL |

**Bits [31:14]**

Reserved, RES0.

**NOOVERFLOW, bit [13]**

When TRCIDR3.NOOVERFLOW == 0b1:

Trace overflow prevention.

<table>
<thead>
<tr>
<th>NOOVERFLOW</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Trace unit buffer overflow prevention is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Trace unit buffer overflow prevention is enabled.</td>
</tr>
</tbody>
</table>

Note that enabling this feature might cause a significant performance impact.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.

**Bits [12:9]**

Reserved, RES0.
ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

<table>
<thead>
<tr>
<th>ISTALL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit must not stall the PE.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit can stall the PE.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally unknown value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from 0b0000 to 0b1111.

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0000</td>
<td>Minimal invasion. This setting has a greater risk of a trace unit buffer overflow.</td>
</tr>
<tr>
<td>0b1111</td>
<td>Maximum invasion. Reduced risk of a trace unit buffer overflow.</td>
</tr>
</tbody>
</table>

Note that for some implementations, invasion might occur at the minimal invasion level.

It is implementation defined whether some of the least significant bits are supported. Arm recommends that bits[3:2] are supported.

On a Trace unit reset, this field resets to an architecturally unknown value.

Accessing the TRCSTALLCTLR

Must be programmed if implemented.

Writes are constrained unpredictable if the trace unit is not in the Idle state.

TRCSTALLCTLR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x02C</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
The TRCSTATR characteristics are:

**Purpose**

Returns the trace unit status.

**Configuration**

External register TRCSTATR bits [31:0] are architecturally mapped to AArch64 System register TRCSTATR[31:0].

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCSTATR are RES0.

**Attributes**

TRCSTATR is a 32-bit register.

**Field descriptions**

The TRCSTATR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>30</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>29</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>28</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>27</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>26</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>25</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>24</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>23</td>
<td>Reserved (RES0)</td>
</tr>
<tr>
<td>22</td>
<td>PMSTABLE</td>
</tr>
<tr>
<td>21</td>
<td>Idle status (IDLE)</td>
</tr>
</tbody>
</table>

**PMSTABLE, bit [1]**

Programmers' model stable.

<table>
<thead>
<tr>
<th>PMSTABLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The programmers' model is not stable.</td>
</tr>
<tr>
<td>0b1</td>
<td>The programmers' model is stable.</td>
</tr>
</tbody>
</table>

This bit is **UNKNOWN** while the trace unit is enabled.

**IDLE, bit [0]**

Idle status.

<table>
<thead>
<tr>
<th>IDLE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit is not idle.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit is idle.</td>
</tr>
</tbody>
</table>

**Accessing the TRCSTATR**

TRCSTATR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x00C</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
Otherwise accesses to this register are RO.
TRCSYNCPR, Synchronization Period Register

The TRCSYNCPR characteristics are:

**Purpose**

Controls how often trace protocol synchronization requests occur.

**Configuration**

External register TRCSYNCPR bits [31:0] are architecturally mapped to AArch64 System register `TRCSYNCPR[31:0]`. RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCSYNCPR are res0.

**Attributes**

TRCSYNCPR is a 32-bit register.

**Field descriptions**

The TRCSYNCPR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RES0 | PERIOD |

**Bits [31:5]**

Reserved, RES0.

**PERIOD, bits [4:0]**

Defines the number of bytes of trace between each periodic trace protocol synchronization request.
**PERIOD** | Meaning
---|---
0b00000 | Trace protocol synchronization is disabled.
0b01000 | Trace protocol synchronization request occurs after $2^8$ bytes of trace.
0b01001 | Trace protocol synchronization request occurs after $2^9$ bytes of trace.
0b01010 | Trace protocol synchronization request occurs after $2^{10}$ bytes of trace.
0b01011 | Trace protocol synchronization request occurs after $2^{11}$ bytes of trace.
0b01100 | Trace protocol synchronization request occurs after $2^{12}$ bytes of trace.
0b01101 | Trace protocol synchronization request occurs after $2^{13}$ bytes of trace.
0b01110 | Trace protocol synchronization request occurs after $2^{14}$ bytes of trace.
0b01111 | Trace protocol synchronization request occurs after $2^{15}$ bytes of trace.
0b10000 | Trace protocol synchronization request occurs after $2^{16}$ bytes of trace.
0b10001 | Trace protocol synchronization request occurs after $2^{17}$ bytes of trace.
0b10010 | Trace protocol synchronization request occurs after $2^{18}$ bytes of trace.
0b10011 | Trace protocol synchronization request occurs after $2^{19}$ bytes of trace.
0b10100 | Trace protocol synchronization request occurs after $2^{20}$ bytes of trace.

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.
- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCSYNCPR**

Must be programmed if TRC1DR3.SYNCPR == 0b0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCSYNCPR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x034</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

**Purpose**

Sets the trace ID for instruction trace.

**Configuration**

External register TRCTRACEIDR bits [31:0] are architecturally mapped to AArch64 System register TRCTRACEIDR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR5.TRACEIDSIZE != 0x00. Otherwise, direct accesses to TRCTRACEIDR are RES0.

**Attributes**

TRCTRACEIDR is a 32-bit register.

**Field descriptions**

The TRCTRACEIDR bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</td>
</tr>
</tbody>
</table>

**Bits [31:7]**

Reserved, RES0.

**TRACEID, bits [6:0]**

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of TRCIDR5.TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCTRACEIDR**

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCTRACEIDR can be accessed through the external debug interface:
This interface is accessible as follows:

- When `!AllowExternalTraceAccess()`, or `OSLockStatus()` or `!IsTraceCorePowered()` accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.
The TRCTSCTRL characteristics are:

**Purpose**

Controls the insertion of global timestamps in the trace stream.

**Configuration**

External register TRCTSCTRL bits [31:0] are architecturally mapped to AArch64 System register TRCTSCTRL[31:0]. RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ETE is implemented and TRCIDR0.TSIZE != 0b00000. Otherwise, direct accesses to TRCTSCTRL are **RES0**.

**Attributes**

TRCTSCTRL is a 32-bit register.

**Field descriptions**

The TRCTSCTRL bit assignments are:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>RES0</td>
</tr>
<tr>
<td>30</td>
<td>EVENT_TYPE</td>
</tr>
<tr>
<td>29</td>
<td>RES0</td>
</tr>
<tr>
<td>28</td>
<td>EVENT_SEL</td>
</tr>
<tr>
<td>27</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>26</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>25</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>24</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>23</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>22</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>21</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>20</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>19</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>18</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>17</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>16</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>15</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>14</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>13</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>12</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>11</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>10</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>9</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>8</td>
<td>Reserved, RES0</td>
</tr>
<tr>
<td>7</td>
<td>EVENT_TYPE, bit [7]</td>
</tr>
</tbody>
</table>

**Bits [31:8]**

Reserved, RES0.

**EVENT_TYPE, bit [7]**

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCTSCTRL.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCTSCTRL.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTRL.EVENT.SEL[4] is RES0.</td>
</tr>
</tbody>
</table>

This bit is RES0 if TRCIDR4.NUMRSPAIR == 0b00000.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Bits [6:5]**

Reserved, RES0.
EVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTLR.EVENT.TYPE controls whether TRCTSCTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

This field is RES0 if TRCIDR4.NUMRSPAIR == 0b0000.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing the TRCTSCTLR

Must be programmed if TRCCONFIGR.TS == 0b1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCTSCTLR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x030</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR characteristics are:

**Purpose**

Controls instruction trace filtering.

**Configuration**

External register TRCVICTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVICTLR[31:0]. RW fields in this register reset to architecturally unknown values.

This register is present only when ETE is implemented. Otherwise, direct accesses to TRCVICTLR are res0.

**Attributes**

TRCVICTLR is a 32-bit register.

**Field descriptions**

The TRCVICTLR bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>RES0</td>
<td>EXLEVEL_NS_EL2</td>
<td>EXLEVEL_NS_EL1</td>
<td>EXLEVEL_NS_EL0</td>
<td>EXLEVEL_S_EL3</td>
<td>EXLEVEL_S_EL2</td>
<td>EXLEVEL_S_EL1</td>
<td>EXLEVEL_S_EL0</td>
<td>RES0</td>
<td>TRCERR</td>
<td>SSSTATUS</td>
<td>RES0</td>
<td>EVENT_TYPE</td>
<td>RES0</td>
<td>EVENT_SEL</td>
</tr>
</tbody>
</table>

**Bits [31:23]**

Reserved, RES0.

**EXLEVEL_NS_EL2, bit [22]**

**When Non-secure EL2 is implemented:**

Filter instruction trace for EL2 in Non-secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_NS_EL2</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The trace unit generates instruction trace for EL2 in Non-secure state.</td>
</tr>
<tr>
<td>01</td>
<td>The trace unit does not generate instruction trace for EL2 in Non-secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally unknown value.

**Otherwise:**

Reserved, RES0.

**EXLEVEL_NS_EL1, bit [21]**

**When Non-secure EL1 is implemented:**

Filter instruction trace for EL1 in Non-secure state.
Meaning

0b0  The trace unit generates instruction trace for EL1 in Non-secure state.
0b1  The trace unit does not generate instruction trace for EL1 in Non-secure state.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [20]

When Non-secure EL0 is implemented:

Filter instruction trace for EL0 in Non-secure state.

Meaning

0b0  The trace unit generates instruction trace for EL0 in Non-secure state.
0b1  The trace unit does not generate instruction trace for EL0 in Non-secure state.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]

When HaveEL(EL3):

Filter instruction trace for EL3 in Secure state.

Meaning

0b0  The trace unit generates instruction trace for EL3 in Secure state.
0b1  The trace unit does not generate instruction trace for EL3 in Secure state.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]

When HaveEL(EL2) and ARMv8.4-SecEL2 is implemented:

Filter instruction trace for EL2 in Secure state.

Meaning

0b0  The trace unit generates instruction trace for EL2 in Secure state.
0b1  The trace unit does not generate instruction trace for EL2 in Secure state.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Otherwise:
Reserved, RES0.

**EXLEVEL_S_EL1, bit [17]**

When Secure EL1 is implemented:

Filter instruction trace for EL1 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL1</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL1 in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL1 in Secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**EXLEVEL_S_EL0, bit [16]**

When Secure EL0 is implemented:

Filter instruction trace for EL0 in Secure state.

<table>
<thead>
<tr>
<th>EXLEVEL_S_EL0</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit generates instruction trace for EL0 in Secure state.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit does not generate instruction trace for EL0 in Secure state.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.

**Bits [15:12]**

Reserved, RES0.

**TRCERR, bit [11]**

When TRCIDR3.TRCERR == 0b1:

Controls the forced tracing of System Error exceptions.

<table>
<thead>
<tr>
<th>TRCERR</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forced tracing of System Error exceptions is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forced tracing of System Error exceptions is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:
Reserved, RES0.
TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

<table>
<thead>
<tr>
<th>TRCRESET</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Forced tracing of PE Resets is disabled.</td>
</tr>
<tr>
<td>0b1</td>
<td>Forced tracing of PE Resets is enabled.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

<table>
<thead>
<tr>
<th>SSSTATUS</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>Stopped State.</td>
</tr>
<tr>
<td></td>
<td>The ViewInst start/stop function is in the stopped state.</td>
</tr>
<tr>
<td>0b1</td>
<td>Started State.</td>
</tr>
<tr>
<td></td>
<td>The ViewInst start/stop function is in the started state.</td>
</tr>
</tbody>
</table>

Before software enables the trace unit, it must write to this bit to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this bit to 0b1. Arm recommends that the value of this bit is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR_SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing TRCPGCTRL.EN or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

This bit is res1 if TRCIDR4.NUMACPAIRS == 0b0000 and TRCIDR4.NUMPC == 0b0000.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, res0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

<table>
<thead>
<tr>
<th>EVENT_TYPE</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>A single Resource Selector. TRCVICTLR.EVENT_SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event.</td>
</tr>
<tr>
<td>0b1</td>
<td>A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT_SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT_SEL[4] is res0.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, res0.
Bits [6:5]

Reserved, RES0.

**EVENT_SEL, bits [4:0]**

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == 0b0000:

This field is reserved:

- Bits [4:1] are RES0.
- Bit [0] is RES1.

Otherwise:

Reserved, RES0.

**Accessing the TRCVICTLR**

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

**TRCVICTLR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x080</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCVIIECTLR, ViewInst Include/Exclude Control Register

The TRCVIIECTLR characteristics are:

**Purpose**

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

**Configuration**

External register TRCVIIECTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVIIECTLR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVIIECTLR are RES0.

**Attributes**

TRCVIIECTLR is a 32-bit register.

**Field descriptions**

The TRCVIIECTLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | EXCLUDE<m>, bit [m+16] | RES0 | INCLUDE<m>, bit [m] |

**Bits [31:24]**

Reserved, RES0.

**EXCLUDE<m>, bit [m+16], for m = 0 to 7**

Selects which Address Range Comparators are in use with the ViewInst exclude function.

Each bit represents an Address Range Comparator, so bit[m] controls the selection of Address Range Comparator m.

<table>
<thead>
<tr>
<th>EXCLUDE&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address range that Address Range Comparator m defines, is not selected for the ViewInst exclude function.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address range that Address Range Comparator m defines, is selected for the ViewInst exclude function.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Bits [15:8]**

Reserved, RES0.
**INCLUDE\(<m>\), bit [m], for m = 0 to 7**

Selects which Address Range Comparators are in use with the ViewInst include function.

Each bit represents an Address Range Comparator, so bit[m] controls the selection of Address Range Comparator m.

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

<table>
<thead>
<tr>
<th>INCLUDE(&lt;m&gt;)</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The address range that Address Range Comparator m defines, is not selected for the ViewInst include function.</td>
</tr>
<tr>
<td>0b1</td>
<td>The address range that Address Range Comparator m defines, is selected for the ViewInst include function.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m \(\geq\) TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCVIIECTLR**

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCVIIECTLR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x084</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCVIPCSSCTLR, ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTLR characteristics are:

Purpose

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

Configuration

External register TRCVIPCSSCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVIPCSSCTLR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMPC > 0b0000. Otherwise, direct accesses to TRCVIPCSSCTLR are RES0.

Attributes

TRCVIPCSSCTLR is a 32-bit register.

Field descriptions

The TRCVIPCSSCTLR bit assignments are:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| RES0 | STOP\langle m\rangle, \text{ bit } [m+16] | RES0 | START\langle m\rangle, \text{ bit } [m] |

Bits [31:24]

Reserved, RES0.

STOP\langle m\rangle, \text{ bit } [m+16], for \( m = 0 \) to 7

Selects which PE Comparator Inputs are in use with ViewInst start/stop function, for the purpose of stopping trace.

<table>
<thead>
<tr>
<th>STOP\langle m\rangle</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0\text{b}0</td>
<td>The PE Comparator Input ( m ), is not selected as a stop resource.</td>
</tr>
<tr>
<td>0\text{b}1</td>
<td>The PE Comparator Input ( m ), is selected as a stop resource.</td>
</tr>
</tbody>
</table>

This bit is RES0 if \( m \geq \text{ TRCIDR4}.\text{NUMPC} \).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

START\langle m\rangle, \text{ bit } [m], for \( m = 0 \) to 7

Selects which PE Comparator Inputs are in use with ViewInst start/stop function, for the purpose of starting trace.
Meaning

<table>
<thead>
<tr>
<th>START&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The PE Comparator Input m, is not selected as a start resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The PE Comparator Input m, is selected as a start resource.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m ≥ TRCIDR4.NUMPC.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCVIPCSSCTRLR**

Must be programmed if TRCIDR4.NUMPC != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCVIPCSSCTRLR can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x08C</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
**TRCVISSCTLR, ViewInst Start/Stop Control Register**

The TRCVISSCTLR characteristics are:

**Purpose**

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

**Configuration**

External register TRCVISSCTLR bits [31:0] are architecturally mapped to AArch64 System register TRCVISSCTLR[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVISSCTLR are RES0.

**Attributes**

TRCVISSCTLR is a 32-bit register.

**Field descriptions**

The TRCVISSCTLR bit assignments are:

<table>
<thead>
<tr>
<th></th>
<th>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</th>
<th>START&lt;&gt;&lt; m&gt;, bit [m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>30</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>29</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>28</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>27</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>26</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>25</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>24</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>23</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>22</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>21</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>20</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>19</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>18</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>17</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>16</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>15</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>14</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>13</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>12</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>11</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>10</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>9</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>8</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>7</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>6</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>5</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>4</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>3</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>2</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>1</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
<tr>
<td>0</td>
<td>STOP&lt;&gt;&lt; m&gt;, bit [m+16]</td>
<td>START&lt;&gt;&lt; m&gt;, bit [m]</td>
</tr>
</tbody>
</table>

**STOP<>< m>, bit [m+16], for m = 0 to 15**

Selects which Single Address Comparators are in use with ViewInst start/stop function, for the purpose of stopping trace.

<table>
<thead>
<tr>
<th>STOP&lt;&gt;&lt; m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single Address Comparator m, is not selected as a stop resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single Address Comparator m, is selected as a stop resource.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= 2 × TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**START<>< m>, bit [m], for m = 0 to 15**

Selects which Single Address Comparators are in use with ViewInst start/stop function, for the purpose of starting trace.

<table>
<thead>
<tr>
<th>START&lt;&gt;&lt; m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The Single Address Comparator m, is not selected as a start resource.</td>
</tr>
<tr>
<td>0b1</td>
<td>The Single Address Comparator m, is selected as a start resource.</td>
</tr>
</tbody>
</table>

This bit is RES0 if m >= 2 × TRCIDR4.NUMACPAIRS.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Accessing the TRCVISSCTLR

Must be programmed if TRCIDR4.NUMACPAIRS > 0b0000.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are constrained unpredictable if the trace unit is not in the Idle state.

TRCVISSCTLR can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x088</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCVMIDCCTLR0, Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCCTLR0 characteristics are:

**Purpose**

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<\text{n}> registers, where \text{n}=0-3.

**Configuration**

External register TRCVMIDCCTLR0 bits [31:0] are architecturally mapped to AArch64 System register TRCVMIDCCTLR0[31:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented, TRCIDR4.NUMVMIDC > 0x0 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTLR0 are RES0.

**Attributes**

TRCVMIDCCTLR0 is a 32-bit register.

**Field descriptions**

The TRCVMIDCCTLR0 bit assignments are:

<table>
<thead>
<tr>
<th>Bit assignment</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP3&lt;m&gt;, bit [m+24]</td>
<td>When TRCIDR4.NUMVMIDC &gt; 3:</td>
</tr>
<tr>
<td>COMP2&lt;m&gt;, bit [m+16]</td>
<td>The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>COMP1&lt;m&gt;, bit [m+8]</td>
<td>The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>COMP0&lt;m&gt;, bit [m]</td>
<td>This bit is RES0 if m &gt;= TRCIDR2.VMIDSIZE.</td>
</tr>
</tbody>
</table>

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Otherwise:**

Reserved, RES0.
When TRCIDR4.NUMVMIDC > 2:

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

<table>
<thead>
<tr>
<th>COMP2&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR2([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR2([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq \text{TRCIDR2.VMIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, res0.

When TRCIDR4.NUMVMIDC > 1:

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

<table>
<thead>
<tr>
<th>COMP1&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR1([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR1([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq \text{TRCIDR2.VMIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, res0.

When TRCIDR4.NUMVMIDC > 0:

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

<table>
<thead>
<tr>
<th>COMP0&lt;(m)&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR0([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR0([(m\times8+7):(m\times8)]) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is res0 if \(m \geq \text{TRCIDR2.VMIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.
Otherwise:

Reserved, RES0.

Accessing the TRCVMIDCCTRL0

If software uses the TRCVMIDCVR<n> registers, where n=0-3, then it must program this register.

If software sets a mask bit to 0b1 then it must program the relevant byte in TRCVMIDCVR<n> to 0x00.

If any bit is 0b1 and the relevant byte in TRCVMIDCVR<n> is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVMIDCCTRL0 can be accessed through the external debug interface:

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x688</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
TRCVMIDCCTLR1, Virtual Context Identifier Comparator Control Register 1

The TRCVMIDCCTLR1 characteristics are:

**Purpose**

Virtual Context Identifier Comparator mask values for the TRCVMIDCVR<n> registers, where n=4-7.

**Configuration**

External register TRCVMIDCCTLR1 bits [31:0] are architecturally mapped to AArch64 System register TRCVMIDCCTLR1[31:0].

RW fields in this register reset to architecturally **UNKNOWN** values.

This register is present only when ETE is implemented, TRCIDR4.NUMVMIDC > 0x4 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTLR1 are **RES0**.

**Attributes**

TRCVMIDCCTLR1 is a 32-bit register.

**Field descriptions**

The TRCVMIDCCTLR1 bit assignments are:

<table>
<thead>
<tr>
<th>31</th>
<th>30</th>
<th>29</th>
<th>28</th>
<th>27</th>
<th>26</th>
<th>25</th>
<th>24</th>
<th>23</th>
<th>22</th>
<th>21</th>
<th>20</th>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMP7&lt;m&gt;, bit [m+24]</td>
<td>COMP6&lt;m&gt;, bit [m+16]</td>
<td>COMP5&lt;m&gt;, bit [m+8]</td>
<td>COMP4&lt;m&gt;, bit [m]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**COMP7<m>, bit [m+24], for m = 0 to 7**

When TRCIDR4.NUMVMIDC > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

<table>
<thead>
<tr>
<th>COMP7&lt;m&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b0</td>
<td>The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>0b1</td>
<td>The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is **RES0** if m >= TRCIDR2.VMIDSIZE.

On a Trace unit reset, this field resets to an architecturally **UNKNOWN** value.

**Otherwise:**

Reserved, **RES0**.
COMP6<\texttt{m}>, bit \([\texttt{m+16}]\), for \(\texttt{m} = 0\) to \(7\)

When \texttt{TRCIDR4.NUMVMIDC} > 6:

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

<table>
<thead>
<tr>
<th>COMP6&lt;\texttt{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>The trace unit includes TRCVMIDCVR6((\texttt{m}×8+7):(\texttt{m}×8)) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>The trace unit ignores TRCVMIDCVR6((\texttt{m}×8+7):(\texttt{m}×8)) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \(\texttt{m} \geq \texttt{TRCIDR2.VMIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

COMP5<\texttt{m}>, bit \([\texttt{m+8}]\), for \(\texttt{m} = 0\) to \(7\)

When \texttt{TRCIDR4.NUMVMIDC} > 5:

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

<table>
<thead>
<tr>
<th>COMP5&lt;\texttt{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>The trace unit includes TRCVMIDCVR5((\texttt{m}×8+7):(\texttt{m}×8)) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>The trace unit ignores TRCVMIDCVR5((\texttt{m}×8+7):(\texttt{m}×8)) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \(\texttt{m} \geq \texttt{TRCIDR2.VMIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.

Otherwise:

Reserved, \texttt{RES0}.

COMP4<\texttt{m}>, bit \([\texttt{m}]\), for \(\texttt{m} = 0\) to \(7\)

When \texttt{TRCIDR4.NUMVMIDC} > 4:

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

<table>
<thead>
<tr>
<th>COMP4&lt;\texttt{m}&gt;</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{0b0}</td>
<td>The trace unit includes TRCVMIDCVR4((\texttt{m}×8+7):(\texttt{m}×8)) when it performs the Virtual context identifier comparison.</td>
</tr>
<tr>
<td>\texttt{0b1}</td>
<td>The trace unit ignores TRCVMIDCVR4((\texttt{m}×8+7):(\texttt{m}×8)) when it performs the Virtual context identifier comparison.</td>
</tr>
</tbody>
</table>

This bit is \texttt{RES0} if \(\texttt{m} \geq \texttt{TRCIDR2.VMIDSIZE}\).

On a Trace unit reset, this field resets to an architecturally \texttt{UNKNOWN} value.
Otherwise:

Reserved, RES0.

### Accessing the TRCVMIDCCTRLR1

If software uses the TRCVMIDCVR<\text{n}> registers, where n=4-7, then it must program this register.

If software sets a mask bit to \text{0b1} then it must program the relevant byte in TRCVMIDCVR<\text{n}> to \text{0x00}.

If any bit is \text{0b1} and the relevant byte in TRCVMIDCVR<\text{n}> is not \text{0x00}, the behavior of the Virtual Context Identifier Comparator is CONSTRANGED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRANGED UNPREDICTABLE if the trace unit is not in the Idle state.

**TRCVMIDCCTRLR1 can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x68C</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:

- When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are RW.
The TRCVMIDCVR\(<n>\) characteristics are:

**Purpose**

Contains the Virtual Context Identifier Comparator value.

**Configuration**

External register TRCVMIDCVR\(<n>\) bits [63:0] are architecturally mapped to AArch64 System register TRCVMIDCVR\(<n>\)[63:0].

RW fields in this register reset to architecturally UNKNOWN values.

This register is present only when ETE is implemented and TRCIDR4.NUMVMIDC > \(n\). Otherwise, direct accesses to TRCVMIDCVR\(<n>\) are RES0.

**Attributes**

TRCVMIDCVR\(<n>\) is a 64-bit register.

**Field descriptions**

The TRCVMIDCVR\(<n>\) bit assignments are:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9  | 8  | 7  | 6  | 5  | 4  | 3  | 2  | 1  | 0  |
| VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE | VALUE |

**VALUE, bits [63:0]**

Virtual context identifier value. The width of this field is indicated by TRCIDR2.VMIDSIZE. Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier.

On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

**Accessing the TRCVMIDCVR\(<n>\)**

Must be programmed if any of the following are true:

- TRCRSCTL\(<a>\).GROUP == \(0b0111\) and TRCRSCTL\(<a>\).VMID[\(n\)] == 0b1.
- TRCACAT\(<a>\).CONTEXT TYPE == \(0b1\) or \(0b11\) and TRCACAT\(<a>\).CONTEXT == \(n\).

**TRCVMIDCVR\(<n>\) can be accessed through the external debug interface:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETE</td>
<td>0x640 + 8(n)</td>
</tr>
</tbody>
</table>

This interface is accessible as follows:
• When !AllowExternalTraceAccess(), or OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
• Otherwise accesses to this register are RW.